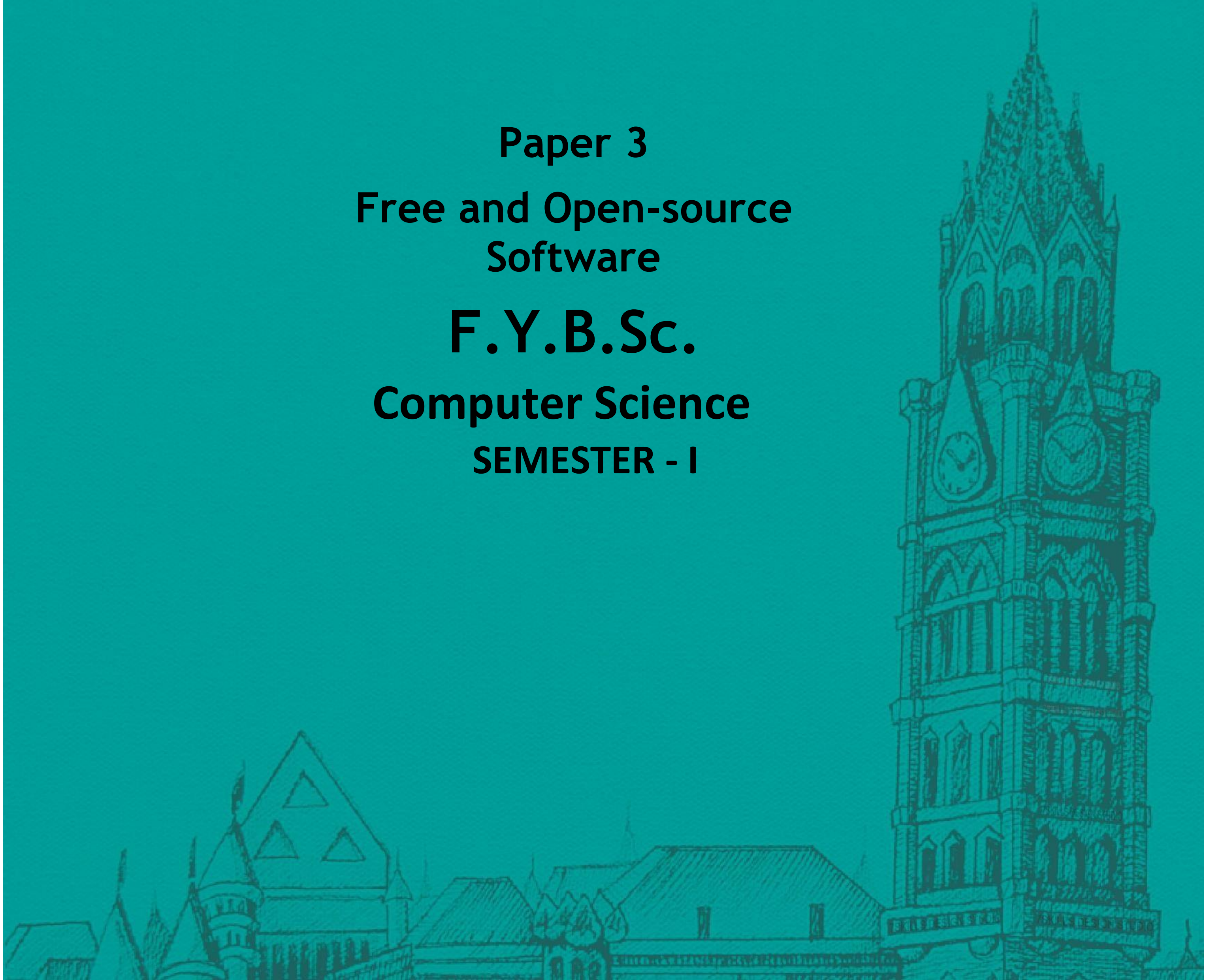
University of Mumbai

INSTITUTE OF DISTANCE AND OPEN LEARNING



Paper 3

Free and Open-source Software

F.Y.B.Sc.

Computer Science

SEMESTER - I

Syllabus

Semester I – Theory

Course: **USCS103**	**Free and Open-source Software** **(Credits : 2 Lectures/Week: 3)**	
Objective: Open Source has acquired a prominent place in software industry. Having knowledge of Open Source and its related technologies is an essential for Computer Science student. This course introduces Open Source methodologies and ecosystem to students. **Expected Learning Outcome:** 1) Upon completion of this course, students should have a good working knowledge of Open Source ecosystem, its use, impact and importance. 2) This course shall help student to learn Open Source methodologies, case studies with real life examples.		
Unit I	**Introduction** Introduction: Open Source, Free Software, Free Software vs. Open Source software, Public Domain Software, FOSS does not mean no cost. History: BSD, The Free Software Foundation and the GNU Project. **Methodologies** Open Source History, Initiatives, Principle and methodologies. Philosophy: Software Freedom, Open Source Development Model Licenses and Patents: What Is A License, Important FOSS Licenses (Apache,BSD,GPL, LGPL), copyrights and copy lefts, Patents Economics of FOSS : Zero Marginal Cost, Income-generation opportunities, Problems with traditional commercial software, Internationalization	**15L**

	Social Impact Open source vs. closed source, Open source government, Open source ethics. Social and Financial impacts of open source technology, Shared software, Shared source, Open Source in Government.	
Unit II	**Case Studies** Example Projects: Apache web server, GNU/Linux, Android, Mozilla (Firefox), Wikipedia, Drupal, wordpress, GCC, GDB, github, Open Office. Study: Understanding the developmental models, licensings, mode of funding,commercial/non-commercial use. Open Source Hardware, Open Source Design, Open source Teaching. Open source media. **Collaboration, Community and Communication** **Contributing to Open Source Projects** Introduction to github, interacting with the community on github, Communication and etiquette, testing open source code, reporting issues, contributing code. Introduction to wikipedia, contributing to Wikipedia Or contributing to any prominent open source project of student's choice. Starting and Maintaining own Open Source Project.	**15L**
Unit III	**Understanding Open Source Ecosystem** Open Source Operating Systems: GNU/Linux, Android, Free BSD, Open Solaris. Open Source Hardware, Virtualization Technologies, Containerization Technologies: Docker, Development tools, IDEs, debuggers, Programming languages, LAMP, Open Source database technologies	**15L**

Text books:

1. Unix Concepts and Applications by Sumitabha Das, Tata McGraw Hill Education, 2006
2. The official Ubuntu Book, 8th Edition

Additional references:

1. The Linux Documentation Project: http://www.tldp.org/
2. Docker Project Home: http://www.docker.com
3. Linux kernel Home: http://kernel.org
4. Open Source Initiative: https://opensource.org/
5. Linux Documentation Project: http://www.tldp.org/
6. Wikipedia: https://en.wikipedia.org/
7. https://en.wikipedia.org/wiki/Wikipedia:Contributing_to_Wikipedia
8. Github: https://help.github.com/
9. The Linux Foundation: http://www.linuxfoundation.org/

Semester I – Practical

USCSP103	**Free and Open Source Software** 1. Identify any Open Source software and create detailed report about it. Sample Guidelines. a. Idea b. What problem does it solves? c. Licensing model d. Intent behind making it open source e. Monetization models f. Popularity g. Impact 2. Learn at least three different open source licenses and create a brief report about them. a. History of license b. Idea c. What problems does it solve? d. Detailed licensing model e. Which popular software are released under this license? f. Any popular news associated with this license? g. Popularity h. Impact 3. Contributing to Open Source a. Identify any Open Source project of your interest b. Learn more about the project w.r.t. Lab1. c. Start contributing to the project either by i. Testing ii. Reporting bugs iii. Coding iv. Helping in documentation v. Participating in discussions vi. Participating in pre-release testing programs vii. UI development. viii. Or any other important area. 4. Hands on with Open Source Software a. Identify any open source software of your interest b. Learn it from practical view-point c. Give a brief presentation about it to the class d. Sample projects: gcc, gdb, drupal, wordpress, apache web server, mysql database 5. Contributing to Wikipedia: a. Introduction to wikipedia: operating model, license, how to contribute? b. Create your user account on wikipedia c. Identify any topic of your choice and contribute the missing information 6. Github a. Create and publish your own open source project:	

Write any simple program using your choice of programming language.

   b. Create a repository on github and save versions of your project. You'll learn about the staging area, committing your code, branching, and merging,
   c. Using GitHub to Collaborate: Get practice using GitHub or other remote repositories to share your changes with others and collaborate on multi-developer projects. You'll learn how to make and review a pull request on GitHub.
   d. Contribute to a Live Project: Students will publish a repository containing their reflections from the course and submit a pull request.

7. Open Source Operating Systems
   a. Learn any open source operating system of your choice : Linux, Android, FreeBSD, Open Solaris etc.
   b. Learn the installation.
   c. Identify the unique features of the OS of your choice.
8. Virtualization: Open Source virtualization technologies:
   a. Install and configure any one: VirtualBox, Zen, KVM
   b. Create and use virtual machines
9. Containerization:
   a. Containerization technologies: docker, rocket, LXD
   b. Install and configure any containerization technology
   c. Create and use containers using it
10. Linux Kernel: Learn Linux kernel with respect to:
    a. What is Linux kernel?
    b. Operating model
    c. Licensing Model
    d. How development works?
    e. Download kernel source code.
    b. Compile the Kernel

Unit -1

INTRODUCTION TO FREE AND OPEN SOURCE SOFTWARE

Unit Structure

1.0 ABSTRACT

Open source software is the software in which users have the ability to run, copy, distribute, study, change, share and improve for any purpose. Open source library software's doesn't need the initial cost of commercial software and enables libraries to have greater control over their working environment. Library professionals should be aware of the advantages of open source software with involvement of its development. One should have basic knowledge about the selection, installation and maintenance. Open source software requires a greater degree of computing responsibility than commercial software. Library professionals do not think seriously about the advantages of open source software for automation and hence are reluctant to use it. They do not have the expertise to support open source software.

1.1 INTRODUCTION

A software for which source code is freely available with a license to study, change and further distributed to any other individual for any purpose is called open source software. Open Source Software is something which you can modify as per your needs, share with others without any licensing violation burden. When we say Open Source, source code of software is available publicly with Open Source licenses which allows you to edit source code and distribute it.

The key fact that makes open source software (OSS) different from proprietary software is its license. As copyright material, software is almost always licensed. The license indicates how the software may be used. OSS is unique in that it is always released under a license that has been certified to meet the criteria of the Open Source Definition. In contrast, creators of proprietary software usually do not make their source code available to others to modify.

Open source software is unique in that it is always released under a license that allows users to access, modify and redistribute the source code. Source code is a specialized language that allows software developers to create and modify computer programs. If you do not have legal access to the source code, then the program cannot be changed or moved to a different kind of computer

1.2 FREE SOFTWARE

Free software means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Free software may be packaged and distributed for a free; the "free" refers to the ability to reuse it, modified or unmodified, as part of another software package. As part of the ability to modify, users of free software may also have access to and study the source code. Thus, “free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer”. We sometimes call it “libre software,” borrowing the French or Spanish word for “free” as in freedom, to show we do not mean the software is gratis.

The concept of free software is the brainchild of Richard Stallman, head of the GNU Project in 1985. He meant Free as in freedom. Because the word free in English means without cost the terms open source was created. The best known example of free software is Linux, an operating system that is proposed as an alternative to Windows or other proprietary operating systems. Debian is an example of a distributor of a Linux package.

Free software is easily confused with freeware, a term describing software that can be freely downloaded and used but which may contain restrictions for modification and reuse.

The four essential freedoms of Free Software

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.
- The freedom to study how the program works, and change it so it does your computing as you wish. Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help others.
- The freedom to distribute copies of your modified versions to others. 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.

1.3 FOSS DOES NOT MEAN NO COST

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.

Free software does not mean non-commercial. A free program must be available for commercial use, commercial development, and commercial distribution. Commercial development of free software is no longer unusual; such free commercial software is very important. You may have paid money to get copies of free software, 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.

A free program must offer the four freedoms to any user that obtains a copy of the software, provided the user has complied thus far with the conditions of the free license covering the software. Putting some of the freedoms off limits to some users, or requiring that users pay, in money or in kind, to exercise them, is tantamount to not granting the freedoms in question, and thus renders the program nonfree.

Most important thing to keep in mind that Free Software is a matter of liberty, not price. While OSS is usually free there are some exceptions. You will usually be able to determine what these exceptions are by considering the total cost of ownership (TCO) involved in adopting and managing open source software. While the software itself may be free, make sure you consider the need for additional services or products, as these may have costs attached (e.g. access to software updates, support services). You also have to take into account possible switching costs. These costs would include moving data from an old system to new systems, training costs and costs involved when switching from one platform to another one (e.g. the costs of switching from Microsoft Windows to a Linux operating system). If your business does not have enough information technology expertise, you may have to outsource outside technical services to provide open source support or to manage its implementation and delivery.

1.4 FREE SOFTWARE VS. OPEN SOURCE SOFTWARE

“Free” and “open source” are two terms commonly used interchangeably in the software industry. Yet, for many developers, the difference between the two is not always clear. This can lead to confusion about how to use each source code, as well as how to make source code available for others.

Difference between Free Software and Open Source Software:

Free Software	**Open Source Software**
Free software usually refers open source under GNU GPL license. Because the word free in English means without cost the terms open source was created.	Your source code is accessible to anyone to read and modify and redistribute depending on license conditions. Publishing source code online without the public being able to modify them doesn’t make lots of sense.
Software is an important part of people’s lives.	Software is just software. There are no ethics associated directly to it.
Software freedom translates to	Ethics are to be associated to the

social freedom.	people not to the software.
Freedom is a value that is more important than any economical advantage.	Freedom is not an absolute concept. Freedom should be allowed, not imposed.
Examples: The Free Software Directory maintains a large database of free-software packages. Some of the best-known examples include the Linux kernel, the BSD and Linux operating systems, the GNU Compiler Collection and C library; the MySQL relational database; the Apache web server; and the Sendmail mail transport agent.	Examples: Prime examples of open-source products are the Apache HTTP Server, the e-commerce platform osCommerce, internet browsers Mozilla Firefox and Chromium (the project where the vast majority of development of the freeware Google Chrome is done) and the full office suite Libre Office.

1.5 PUBLIC DOMAIN SOFTWARE

Public domain software is any software that has no legal, copyright or editing restrictions associated with it. It is free and open-source software that can be publicly modified, distributed or sold without any restrictions. SQLite, I2P and CERN are popular examples of public domain software. As well as Many different items can be labelled as public domain. For instance, books, speeches, poems, artwork, songs, and videos can all be made freely available to the public. In the computing world, "public domain" is often used to refer to software programs that are offered to the public without copyright restrictions.

The copyright protection an item in the public domain may have expired, been released by the author, or never existed in the first place. Public domain software has no ownership and is available for use, modification and commercialization by anyone. Typically, public domain software is intentionally or voluntarily uncopyrighted, unpatented and is unrestricted by its developer/author. It is different from free software and freeware that does has copyrights and patents associated with it.

Although there are no licensing requirements with public domain software, The Unlicensed, Creative Commons License and WTFPL are based on a similar approach.

Public domain software is similar to open source software, in which the source code of a program is made publicly available. However, open source software, while freely distributed, still retains the original developer's copyright. This means the developer can change the redistribution policy at any time. Public domain software is also similar to freeware, which refers to software offered at no charge. However, like open source software, freeware programs are still protected by copyright. Therefore, users may not redistribute the software unless they receive permission from the original developer.

Since there are many similarities between freeware, open source, and public domain software, the terms are often used interchangeably. However, there are important legal differences between the licenses, so it is important for developers to choose the correct license when releasing software programs. Public domain software, which offers the least legal protection, is most often published by individuals or educational institutions, rather than companies. When software is offered as public domain, it is often labelled "PD" or may include a Public Domain Mark (PDM).

1.6 HISTORY OF BSD

Berkeley Software Distribution (BSD) is a prominent version of the Unix operating system that was developed and distributed by the Computer Systems Research Group (CSRG) from the University of California at Berkeley between 1977 and 1995. This operating system was originally made for the PDP-11 and DEC VAX computers.

Historically, BSD has been considered as a branch of UNIX — "BSD UNIX", because it shared the initial codebase and design with the original AT&T UNIX operating system. In the 1980s, BSD was widely adopted by vendors of workstation-class systems in the form of proprietary UNIX variants such as DEC ULTRIX and Sun Microsystems Sun OS. This can be attributed to the ease with which it could be licensed, and the familiarity it found among the founders of many technology companies of this era.

Though these commercial BSD derivatives were largely superseded by the UNIX System V Release 4 and OSF/1 systems in the 1990s (both of which incorporated BSD code), later BSD releases provided a basis for several open source development projects which continue to this day.

Today, the term of "BSD" is often non-specifically used to refer to any of these BSD descendants, e.g. FreeBSD, Net BSD or Open BSD, which together form a branch of the family of Unix-like operating systems.

Arrival of Berkeley's Unix

The year is 1974, and BSD began taking shape when Unix first arrived at the University of California at Berkeley. Ken Thompson took a sabbatical from Bell Labs in 1975 and came to visit his Alma Mater as a visiting professor. During this time he helped install Version 6 Unix and started working on a Pascal implementation.

As students continued working on Pascal and implemented an improved text editor called ex, other universities became interested in the software. So, in 1977 Bill Joy, one of those students, started compiling the first Berkeley Software Distribution, or 1BSD which was released on March 9th of the following year with some 30 copies sent out.

Some well-known software that is still in use today had its start in the next version, 2BSD, such as vi and csh, which saw approximately 75 copies distributed.

In 1978 a new more powerful VAX computer was installed at Berkeley and provided a new target for BSD software. Over time large parts of the Operating

System had to be replaced, for instance the initial Unix port to the VAX architecture did not take advantage of the VAX's virtual memory capabilities so much of the kernel was rewritten. The release of 3BSD in 1979 contained this new kernel and ports of the other BSD programs to the VAX architecture.

As BSD spread to more and more institutions, users began adding additional functionality and programs, and sending those back to the team at Berkeley to be included in the next release of BSD. This was the start of the open source movement, before it had a name.

In 1989 a new release called Network Release 1 or Net/1 was made under the BSD license. This contained work that was done on implementing the OSI network protocol stack and new TCP/IP algorithms. It was motivated by the increasing cost of AT&T software licenses and several groups had started to express interest in a separate release of just the network code.

After Net/1 Keith Bostic proposed that more of the system be released under the BSD license and so he lead a project to reemployment most of the standard Unix utilities without any AT&T code.

Within the next 18 months, all of the AT&T utilities had been rewritten and only a few AT&T files remained in the kernel. In 1991 Network Release 2 or Net/2 was made available without those files, resulting in nearly a complete operating system that was freely distributable.

In 1992, Bill and Lynne Jolitz, both Berkeley alumni, release 386BSD 0.0, the first version of BSD for the Intel 386, a computer many had in their homes. This was made possible by Keith Bostic and partially influenced by Richard Stallman.

1.7 THE FREE SOFTWARE FOUNDATION AND THE GNU PROJECT

The FSF

The FSF (*Free Software Foundation*) was founded in the early eighties by Richard M. Stallman, researcher at MIT's Artificial Intelligence laboratory. The foundation's objective is to develop *free* software. That is software that you can freely copy, use, modify, and redistribute as you wish. The only condition is that the source code of these programs must be freely available on demand.

It is important to understand that the term *Free* in *Free Software Foundation* does not refer to *price*, but to *freedom*. These programs can be bought and sold, but there is always a legal way to obtain them gratis.

The GPL

The GPL (*General Public License*) specifies the conditions under which all GNU software is ditributed. The LGPL (Library General Public License) was the corresponding license used for sub-program libraries (please see Why you shouldn't use the Library GPL for your next library for an explanation). The GNU Lesser General Public License is the new replacement for the LGPL.

Roughly, these licenses specify that GNU software may be copied, modified, and redistributed in any manner as long as the source code remains freely available.

The main advantage of software distributed under these conditions is that if you wish to improve the program, you may; and you may redistribute your new and improved version. Thus, everyone benefits. This leads to programs of excellent quality, written by dozens of different people.

The GNU Project

The FSF's GNU (*GNU is not Unix*) Project's objective is to develop a complete operating system, distributed under the conditions of the GPL. This operating system uses some UNIX concepts, but is not UNIX. Richard Stallman started this project on his own right after he founded the FSF. The first part of the project was to program the editor with which he could then program the rest of the software. That editor is the now famous GNU Emacs. He then wrote a C compiler to compile his operating system. That would be the famous GCC. Since then, many people have joined him to write all sorts of programs. The operating system itself, known as HURD, has recently become available.

In addition to the main GNU programs, there are GNU versions of most of the UNIX utilitaires. The GNU versions are often more powerful and reliable than their proprietary counterparts.

As interest in using Emacs was growing, other people became involved in the GNU project, and we decided that it was time to seek funding once again. So in 1985 we created the Free Software Foundation (FSF), a tax-exempt charity for free software development. The FSF also took over the Emacs tape distribution business; later it extended this by adding other free software (both GNU and non-GNU) to the tape, and by selling free manuals as well.

Most of the FSF's income used to come from sales of copies of free software and of other related services (CD-ROMs of source code, CD-ROMs with binaries, nicely printed manuals, all with the freedom to redistribute and modify), and Deluxe Distributions (distributions for which we built the whole collection of software for the customer's choice of platform). Today the FSF still sells manuals and other gear, but it gets the bulk of its funding from members' dues. You can join the FSF at fsf.org.

Free Software Foundation employees have written and maintained a number of GNU software packages. Two notable ones are the C library and the shell. The GNU C library is what every program running on a GNU/Linux system uses to communicate with Linux. It was developed by a member of the Free Software Foundation staff, Roland McGrath. The shell used on most GNU/Linux systems is BASH, the Bourne Again Shell(1), which was developed by FSF employee Brian Fox.

We funded development of these programs because the GNU Project was not just about tools or a development environment. Our goal was a complete operating system, and these programs were needed for that goal.

1.8 SUMMARY

In this chapter we learned, software remains free of charge and they make money by helping others to install, use and troubleshoot it. Free and OSS development is emerging as an alternative approach for developing large software systems. New types and new kinds of software processes are emerging within F/OSSD projects, as well as new characteristics for development project success, when compared to those found in traditional industrial software projects and those portrayed in software engineering textbooks. As a result, F/OSSD offers new types and new kinds of processes to research, understand, improve, and practice.

1.9 REFERENCES

- https://www.howtogeek.com/129967/htg-explains-what-is-open-source-software-and-why-you-should-care/
- https://www.synopsys.com/glossary/what-is-open-source-software.html
- https://dwheeler.com/oss_fs_eval.html
- https://opensource.com/resources/what-open-source
- https://ifap.ru/library/book105.pdf
- Monika Sharma, learn about the social and financial impacts of Open Source, collaborative Organization flourishing in recent years. on February 18, 2018
- Bhattacharya, I, Sharma, K (2007). India in the knowledge economy
- https://opensource.org/history

❖❖❖❖

Unit -2

METHODOLOGIES OF FREE AND OPEN SOURCE SOFTWARE

Unit Structure

2.0 OPEN SOURCE HISTORY

The open source movement is based on a radical retake on copyright law to create high quality software whose use and development are guaranteed to the public. In this article we trace the history of the movement, highlighting its interaction with intellectual property law. The movement has spawned open source software (OSS) communities where developers and users meet to create software that meets their needs. We discuss the demographic profile of OSS participants, their ideology, their motivations, and the process of OSS development. Then we examine the impacts of OSS on society as a whole from the perspective of the information society, discussing the effects on OSS developers, users of OSS, and society at large, particularly in developing countries.

Free software (later renamed "open source software") appeared even before people started thinking in terms of proprietary software, at a time when software development was ruled by open source principles. In the 1960s and 1970s, software programming was mainly performed in both academic and corporate laboratories by scientists and engineers who freely gave, exchanged and modified software. In the early 80s, as software programming increasingly turned proprietary, Richard Stallman founded the Free Software Foundation (FSF) to define and 4 diffuse legal mechanisms and conceptual principles of what he called "free software". By writing the GNU Manifesto (Stallman, 1985), he communicated his ideological view of the nature of intellectual property rights as regards software, and started attracting convinced developers to join him in his GNU Project (GNU stands for "GNU's not

UNIX”). In 1989, the FSF released the GNU General Public License (GPL) version 1 (the updated version 2 was released in 1991) which legalizes copyleft mechanisms and grants end-user’s freedoms in software copies and derivative works.

Turning copyright around “Copyleft” as expressed by the GPL has had a critical effect on shaping the very existence of open source software communities. Open source software uses copyright law to preserve certain freedoms (hence the name, “free software”) regarding the creation, modification, and sharing of software. Specifically, all open source software grants users the following key rights:

1. The right to full access to the source code. When a computer programmer sees how a piece of software actually works, as specified in the source code, they can fully understand the inner workings and can intelligently modify the software as they deem appropriate

2. The right for anyone to run the program for any purpose without restriction. There are no restrictions against commercial, military, foreign, or any other use, and discrimination against users for any reason is expressly forbidden.

3. The right to modify the source code. This includes absorbing the software, in whole or in part, into other pieces of software created by other developers.

4. The right to distribute both the original software and the modified software. A key difference between “free software” and “freeware” is that while freeware generally permits 5 and encourages free distribution of the software, it does not permit sale of the distributed software beyond reasonable distribution costs; free software, in contrast, permits resale at any price.

5. The right to know about their open source rights: The open source license must be prominently displayed and distributed to users, so that they are aware of their rights (including access to the source code). Practically, since users are aware that they can obtain the source code for free, the sale price of OSS tends to be zero, or quite low.

While the preceding five rights constitute open source software, the FSF’s GPL, the first legal document to license open source software, goes further. The GPL grants users and developers these rights with the intention that developers would modify the software and share it with others with similar liberality, and in accordance with Stallman’s personal beliefs on the ethical rightness of sharing software, the GPL assures sharing by further incorporating the concept of “copyleft”. Copyleft is an obligation that the distributor of OSS agrees to in order to receive the privileges mentioned above:

6. The obligation to distribute derivatives under copyleft. Any software modified under the GPL can be redistributed for sale, but it must be licensed under a copyleft license; that is, modified derivative works must also be made available under an open source license. While it does not have to be licensed under the GPL itself, the chosen distribution license may not restrict any of the five rights listed above.

These copyleft terms are critical to the very existence of OSS communities. When Richard Stallman posted his manifesto and invited software developers to join him in his crusade for free software, there was no lack of sympathetic and willing hackers who wanted a return to the days of free sharing. However, there was a grave concern that, corporate interests could 6 easily take these programs, add their proprietary extensions, and withdraw the software from public access. With its copyleft mechanism, the GPL guaranteed that any person or corporation who wanted to benefit from the liberal efforts of computer programmers would be legally bound to share their work in the same spirit of camaraderie. Considering the climate in which the free software movement was founded, it is unlikely that the movement could have gotten off the ground without such a radical clarion call to mobilize devoted followers in the first place.

2.1 INITIATIVES

In 1991, Linus Torvalds, a 21-year-old Finnish programmer, started the famous Linux Project, released under the GPL, which implements a Unix-like operating system on Intel-based microcomputers. The project has grown rapidly to produce a powerful, fast, efficient, stable, reliable, and scalable operating system. The number of Linux users and developers has expanded rapidly. Not including free downloads and installations of Linux, "25% of servers and 2.8% of desktop computers ran paid distributions of Linux as of 2002. The Linux market is rapidly growing and is projected to exceed $35.7 billion by 2008". Moreover, version 2.2.10 of the kernel lists 190 names, though the total number of contributors was estimated at around 1,200.

A major paradigm shift occurred in 1998, when Bruce Perens and Eric Raymond expressed their suspicion that firms may not be convinced by GPL due to Stallman's term, "free" software, "which might understandably have an ominous ring to the ears of business people". The "open source" software movement was created based on the integrating of all the previous licensing that had been prior designed. As a result, a major chasm appeared between Stallman's free software view, which was more ideological and philosophical, and Perens/Raymond's open source view, whose purpose was more business-oriented. The latter was 7 aimed at fighting against the evil "proprietary software" by convincing firms to rely on OSS.

Several major OSS projects have marked people's mind in such software revolution. The Apache web-server project started in early 1995 and has become the most popular Web server software, controlling over 60 percent of the market, much more than Microsoft and Netscape both combined. Inspired by Eric Raymond's paper "The Cathedral and the Bazaar". Netscape, one of the main actors in the Internet browser industry, decided to release its source code by creating Mozilla, its first open source version of its former Communicator. In November 2004, the Mozilla Foundation announced the worldwide availability of the Mozilla Firefox 1.0 web browser.

Today, all major hardware and software vendors have at least forayed into the open source approach. For example, Apple Computer surprisingly followed this model in 2000 when they released the kernel of their Unix-based Mac OS X operating system to the open source community as Darwin 1.0. In 2005, IBM announced their plan to spend $100 million over the next three years to build support for Linux into desktop applications for its Workplace software. Meanwhile, IBM has planned to spread Linux worldwide. In 2004, IBM concentrated on implementing Linux in Brazil, Russia, India and China. Since 2005, the company has planned to increase its efforts in those countries but will also begin extend its programs in Eastern Europe.

2.2 PRINCIPLES AND METHODOLOGY OF THE OPEN SOURCE

- **Principles of the open source**

Transparency: Whether we're developing software or solving a business problem, we all have access to the information and materials necessary for doing our best work. And when these materials are accessible, we can build upon each other's ideas and discoveries. We can make more effective decisions and understand how decisions affect us.

Collaboration: When we're free to participate, we can enhance each other's work in unanticipated ways. When we can modify what others have shared, we unlock new possibilities. By initiating new projects together, we can solve problems that no one can solve alone. And when we implement open standards, we enable others to contribute in the future.

Release early and often: Rapid prototypes can lead to rapid discoveries. An iterative approach leads to better solutions faster. When you're free to experiment, you can look at problems in new ways and seek answers in new places. You can learn by doing.

Inclusive meritocracy: Good ideas can come from anywhere, and the best ideas should win. Only by including diverse perspectives in our conversations can we be certain we've identified the best ideas, and decision-makers continually seek those perspectives. We may not operate by consensus, but successful work determines which projects gather support and effort from the community.

Community: Communities form when different people unite around a common purpose. Shared values guide decision making, and community goals supersede individual interests and agendas.

- **Methodology of the open source**

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 extensive peer 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 responsibility for 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 they cost 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 at the 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 be commended 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 interested in and capable of debugging source code (operating system code, if the system in question is Linux), and obviously doesn't scale beyond a small cadre of highly motivated programmers.

By largely ignoring upstream defect removal and emphasizing downstream defect 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 Software 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 a better 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 labor 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 be fully 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 the open-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.

2.3 PHILOSOPHY: SOFTWARE FREEDOM

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 a precondition 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.

- **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 forbidden or 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.

- **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.

- **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 other words, 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.

2.4 OPEN SOURCE DEVELOPMENT MODEL LICENSES AND PATENTS

What is a Licence?

The simplest explanation is that open source licenses are legal and binding contracts between the author and the user of a software component, declaring that the software can be used in commercial applications under specified conditions. The license is what turns code into an open source component. Without an open source license, the software component is unusable by others, even if it has been publicly posted on GitHub.

Each open source license states what users are permitted do with the software components, their obligations, and what they cannot do as per the terms and conditions. This might sound pretty straight forward, but there are over 200 open source licenses out there so good luck keeping them all organized. Varying in complexity and requirements, it is up to organizations to choose which licenses are most compatible with their policies to ensure that they remain compliant.

These licenses are intended to permit, and indeed, to encourage the contributions of others to the project. Nonetheless, one of the first open development projects relied, at least at the beginning, on a relatively small number of closely-knit developers. This project was Richard Stallman’s plan to develop a complete operating system modeled after the Unix operating system but written entirely in free code.

The main problem in this context is that open source licenses are subjective. Their interpretation depends on the technical usage of the licensed software. Therefore, it’s difficult to determine the legal risks of using open source software, especially for developers, who are not usually legal experts. What developers need is a broad classification of licenses based on the risks they pose in terms of legal compliance.

2.5 IMPORTANT FOSS LICENSES

I. GNU General Public License (GPL)

GNU General Public License is the most popular open source license around. Richard Stallman created the GPL to protect the GNU software from becoming proprietary, and it is a specific implementation of his "copyleft" concept.

GPL is a copyleft license. This means that any software that is written based on any GPL component must be released as open source. The result is that any software that uses any GPL open source component (regardless of its percentage in the entire code) is required to release its full source code and all of the rights to modify and distribute the entire code.

There has always been some confusion regarding what constitutes a 'work based on' another work, which in turn triggers the GPL reciprocity obligation. The FSF tried to add more clarity to GPLv3 as to when the reciprocity obligation is triggered. The FSF even wrote a new GPL license, the Affero license, to address a specific confusion referred to as the "ASP loophole".

In addition, the FSF tried to increase the compatibility of the GPLv3 with other licenses. To combine two codes into a larger work, both the programs must permit it. If such rights are granted by both the programs' licenses, they are compatible. By making the GPLv3 more compatible, the FSF expanded development options.

II. The Apache License

The Apache License is an open source software license released by the Apache Software Foundation (ASF). It's a popular and widely deployed license backed by a strong community. The Apache License allows you to freely use, modify, and distribute any Apache licensed product. However, while doing so, you're required to follow the terms of the Apache License.

The Apache Group (later named the Apache Software Foundation) released the first version of its license in 1995, but it's rare that you'll come across components that still carry this license.
In 2000, when Berkeley accepted the argument put to it by the Free Software Foundation and retired their advertising clause from the BSD license and formed the modified BSD license, Apache did likewise and created the Apache License version 1.1.

Removing the advertising clause meant that the advertising materials of the derivative works of any Apache licensed product were no longer required to include the Apache License attribution. It became ok to include the attribution in the documentation alone.

In 2004, the ASF decided to depart from the BSD model a little more radically and produced the Apache License version 2.0 by granting patents rights and defining solid definitions of the concepts it uses to make

III. Berkeley Software Distribution (BSD)

BSD Licenses or the original BSD License and its two variants - the Modified BSD License (3-clause), and the Simplified BSD License/FreeBSD License (2-clause) are a family of permissive free software licenses.

The BSD License lets you freely modify and distribute your software's code in the source or binary format as long as you retain a copy of the copyright notice, list of conditions, and the disclaimer.
The original BSD License or the 4-clause BSD License also contains an advertising clause and a non-endorsement clause (detailed explanation about these clauses are offered in the following questions). The modified BSD License or the 3-clause BSD License was formed by removing the advertising clause from the original BSD License. Further, the FreeBSD version or the 2-clause BSD License was formed by removing the non-endorsement clause from the modified BSD License or the 3-clause BSD License.

IV. Common Development and Distribution License (CDDL)

CDDL is an open source license published by Sun Microsystems to replace the Sun Public License (SPL). The CDDL license is considered by Sun (now Oracle) to be SPL version 2. It is inspired by the Mozilla Public License (MPL). Sun used to release its free software / open source projects under its Sun Public License (SPL) before it turned to rely upon the CDDL in 2004. CDDL is often dubbed as a cleaned up version of the MPL and is made to facilitate reusability.

You're free to reproduce and distribute any original or derivative works of any software licensed under the CDDL. However, you must not remove or make any changes to any copyright, patent or trademark notices contained in the software. You must also retain any notices of licensing or any descriptive text giving attribution to any contributor or the initial developer.

When you distribute your software in an executable form (any form other than source code), you are required to make the source code available as well under the CDDL. The executable form may be released under the CDDL or any CDDL compatible licenses.

The source code that you have to make available includes your contributions as long as they are an addition to, deletion from or modification of the contents of a file containing the original software – or new files that contain parts of the original program. That means that if your additions are made in separate and independent files that do not contain the original code, you do not have to release it under the CDDL. You may do that if you choose to, but you're not obliged.

In addition, you must include a copy of the CDDL with any source code that you distribute. For each modification that you make, you must identify yourself as the modifier by including a notice in your modified files.

V. Eclipse Public License (EPL)

The Eclipse Public License (EPL) is an open source license developed by the Eclipse Foundation. It's derived from the Common Public License (CPL). The Eclipse codebase now available under the EPL was formerly licensed under the CPL.

The EPL license is a copyleft license. If you modify an EPL'ed component and distribute it in the source code form as part of your program, you're required to disclose the modified code under the EPL. If you distribute such a program in its object code form, you're required to state that the source code can be made available to the recipient upon request. You're also required to share the method for requesting the source code.

The Eclipse Foundation makes clear that, in their opinion, 'merely interfacing or interoperating' with an Eclipse plugin does not make your code a derivative work of the plugin.

If you redistribute a program with an EPL component, you are obligated to include the full license text and the copyrights.

The EPL protects the author from possible lawsuits or damages caused if a company used his/her component in a commercial product. It also offers a patent grant.

VI. MIT License

MIT is one of the most permissive free software licenses. Basically, you can do whatever you want with software licensed under the MIT license - only if you add a copy of the original MIT license and copyright notice to it. Its simplicity is the reason behind its high adoption rate among developers.

2.6 COPYRIGHTS AND COPYLEFTS

Copyright
©

Copyright is a bundle of rights granted to a creator providing him with exclusive rights over his original artistic and literary creation. It is not necessary that a copyright be registered, it is attached automatically to any original artistic work. When the idea of a creator is converted to a material form, the same immediately gets protected under the copyright. For a work to be copyrighted, it is necessary that the work is original work of literature, drama, music, or any other art having artistic value. While copyright protects the material form of an idea it does not protect the idea in itself. It is essential and of significant importance, that permission is sought and the same is granted by the copyright owner before it is republished or reproduced.

The bundle of rights granted to the copyright owner includes the rights to reproduce, copy, publish, communicate, and translate the copyrighted work. Such a right is a natural right granted to the owner of the artistic work immediately on the making of the same.

Copylef

🄯

Most of the creative works, including software programs and codes, comes within the domain of copyright protection and therefore can be copyrighted. However, it is to be noted that software and programming is an area where already existing programs many-a-times are used as a base to build new software or program. It is for this reason, many software owners tend to grant a license to its users a license allowing them to modify and alter their work. Such permission and license can be referred to as Copyleft.

Copyleft can be said to be a specific kind of a license that allows free use of copyrighted material but under certain terms and conditions, granted by the owner of the copyright himself. For instance, software having a copylefted license can be modified, used, distributed, or reproduced provided the source code kept open and available to the public. A copylefted software must be transferred with a similar copyleft license to all its successive users and the license also shall mandate any modification to the software shall be copylefted in a similar manner. In simpler words, copyleft is a license that provides original work to a third person giving him certain rights like that of copying and modifying and any new work carved out based on such original work shall have a copyleft license in a similar manner.

The main objective of a copyleft license is to provide people with opportunities to use and modify an original work, and later grant a similar set of rights to all other interested people. Thus any person who receives a copylefted work and then modifies the same, he cannot restrict the rights to himself alone over the modified work.

The concept of copyleft came as a refreshing idea on the free movement of workers. It broadens the area by not just restricting the rights over the work to a single individual or a small group of people but including all the people having an interest in the work and are ready to comply with the conditions of the copyright license agreement.

To conclude Copyleft is an option derived within the domain of copyright laws itself, providing a little bit of freedom and liberty, that allows users to modify and distribute the software and the program, which was surely not possible with the traditionally copyrighted programs. It would not be wrong to conclude that the concept of copyleft will create communal ownership of several individuals within the confines and limitations of copyright laws.

2.7 PATENTS ECONOMICS OF FOSS

Unlike the holder of an Open Source license, the owner of a patent has exclusive rights over the patented software. No one else can make, use, modify, or sell patented software, and the source code is not available to the public.

Patent rights give the holder control over who uses software and for what purpose. Though software developers can protect their work using both copyrights and patents, copyrights only protect the code itself. Patents, however, protect the program's functionality.

Patents are better than copyrights for software developers because they protect the program regardless of the code and language used. In comparison, copyrights aren't very practical for developers. If you want to release Open Source software while retaining some rights, a copyright only gives you power over someone who steals your work verbatim.

The original idea of a patent was to give the innovator who develops the idea a monopoly of time in which she/he can benefit by commercial exploitation of the patent, protect by legal means from other wishing to copy the idea. But long ago, this has become buried in legal, cultural, administrative and practical difficulties – and this is making waves. We have been patents being used (as with copyright) as a means of proxy business-competition – with a recent **Wired article exposing the battle lines of patent-warfare in the smart phone market** as the big player jostle for position – so Apple sues HTC (used in many Android phones), Nokia sues Apple, Kodak sues Apple, Research in Motion (makers of the BlackBerry) & Samsung while Palm and Apple argue over patents:

The intention is to promote the rapid adoption and adaptation of ideas, benefit the inventors and reward the whole process of research and development. However, over the past two decades, changes in the way patents are attributed and patent holders' increasingly aggressive tactics have created a situation that some claim is choking, rather than promoting, innovation. What makes the problem intractable is that today it is impossible to design a high-end tech product that does not include others' patents.

Software patents do not appear to show a strong effect on FOSS developer motivation in general. This is true for both camps in the software patent debate: the presence of software patents has no positive effects on monetary and skills-related motivation, as argued by proponents; it also does not show negative effects on joy- and self-expression-related motivation, as argued by opponents. In contrast and counter-intuitively, joy-related motivation seems to be positively influenced by the presence of software patents.

2.7.1 Zero Marginal Cost

At the core of the financial aspects of Free and Open Source is the zero negligible expense of merchandise in an environment that is digital. Right now, the rise of Free and Open Source speaks to an affirmation of old-style microeconomic value hypothesis - that a marginal cost in an ideal market approximates the minimal expense. From this point of view, Free and Open Source can be comprehended as a pioneer in arriving at what can be comprehended as a developmentally steady powerful Nash balance in a genuinely free market. Marginal cost is the term utilized in the study of financial aspects and business to allude to the increment in the actual development cost coming about because of delivering one extra unit of the product.

While Free and Open Source is allowed to the end client, there is an expense related to building up the product. These expenses might be littler than creating exclusive programming since building up the task under Free and Open Source permit implies that:

Various online interfaces like Source Forge would offer web facilitating, content store, mailing records and other basic highlights for nothing.

The expense of promoting a Free and Open Source venture (like introducing it in the related gatherings) is typically lower.

Creating something under GPL may give free access to top-notch parts (like QT) that are in any case costly to purchase or not accessible by any stretch of the imagination.

All things considered; improvement of any product initially requires the designer time. Without a doubt, extremely famous activities may hope to get an excellent code commitment for nothing.

2.8 INCOME-GENERATION OPPORTUNITIES

While contributing time and exertion in creating, improving and documenting Free and Open Source doesn't give any immediate salary, the improvement of skill in Free and Open Source gives an expansive scope of revenue generation opportunities - from producing in-house investment funds from upgrades to Free and Open Source to counseling openings in installing, preparing, customizing and the arrangement of TechSupport for Free and Open Source establishments. Yochai Benkler furnishes a fantastic investigation - with IBM's strategy as a key model - of ways that salary and riches are being created through open source and open substance techniques.

With IT budgets increasingly strained, more and more companies are looking to open source software to help lower costs. And while many people associate open source with free software, the movement provides resellers and system integrators (SIs) with significant services revenue, analysts say.

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.

2.9 PROBLEMS WITH TRADITIONAL COMMERCIAL SOFTWARE

First, it ought to be noticed that the business software industry is one of the biggest and most significant ventures on the planet. The ascent of the Open Source development doesn't spell the finish of the business software industry. Numerous individuals contend that business software can be strengthened by the utilization of open-source strategies. Business software is intended to give an item worth paying to and its majority is. Regardless of its sticker price, business software is frequently in demand.

Business products will be refreshed much of the time to mirror the fluctuating requests of the market and client needs. These necessities can prompt software rehearses that rework and change the product too as often as possible, or disperse beta forms as business attempts bringing about high paces of bugs in early forms.

Some business programs are over structured and written in messy code prompting enlarged, slow, running projects. Open Source by differentiating is driven by the necessities of the end clients. Along these lines, their code is regularly of a predominant bore than that of software engineers in the business condition. There is likewise generally an extremely broad level of input as the software engineers test their products with a wide system of individuals.

At the point when source code is accessible, it tends to be checked against different "secondary passages" and other security openings that might be deliberately or accidentally left in the closed source programming. In the past, such gaps have been found in different exclusive items, including software, utilized by the legislature. Having source code additionally implies that the product can be effectively ported to run under various processors, gadget or OS.

Another issue with customary business software is its closed nature. There is generally no or restricted opportunity to alter the copyrighted item.

Additionally, organizations frequently power clients to follow update ways that they may not wish to follow. Open-source Software empowers the person to tweak the product to his end needs in all opportunities.

Another vigorously censured part of business software is that clients are regularly secured to an item because to continue utilizing information documents you are frequently compelled to keep on utilizing a similar program. If you wish to impart documents to clients that have overhauled, you frequently need to update yourself or be discounted as unessential. Since open-source programming permits contending projects to share various information record types, there is no motivation to be caught into any one program. If a more current adaptation has another record group, at that point there regularly are converters that permit clients of more established variants to keep up their information documents forward-thinking.

2.10 INTERNATIONALIZATION

Internationalized software must enable easy porting to other locales. A locale defines language and specific cultural conventions.

In the private international law rules for transfer and license contracts has shown that it is hardly possible to anticipate the applicable law for Copyright Assignment CAs or Copyright License Agreement CLAs when it comes to legal disputes. The first source of uncertainty is the lack of internationally accepted principles of private international law. European, US and Japanese courts apply different conflict-of-law rules for the different legal issues raised by CAs/CLAs. A second source of ambiguity is the lack of precise conflict rules in legislation or case law within the analysed jurisdictions with regard to transfer or license contracts. This makes it hard to predict which law will finally be applicable to the legal questions at stake, even if one could anticipate whether a European, US or Japanese court will be called to hear the case. A third source of legal problems relates to the boundaries of party autonomy in international copyright law. For some of the most crucial aspects of CAs/CLAs, e.g. the question of whether copyright or” joint authorship” in a work can be assigned, the territoriality principle precludes any choice of law. Thus, the parties’ latitude to reduce the complexity of their relationship by a contractual choice is restricted.

2.11 SUMMARY

In this chapter we learned that, while open source software offers a multitude of benefits, it introduces a whole new level of code management that does not exist when solely using commercial software. It is critical that an organization utilizing OSS, or acquiring codebases that contain OSS in a merger or acquisition, truly understand what is in their code so they can effectively manage and secure it. The Synopsys solution suite offers complete open source coverage, so you can use OSS confidently. In short, open source provides a way for companies to collaborate on technology that's mutually beneficial.

2.12 REFERENCES

- Monika Sharma, learn about the social and financial impacts of Open Source, collaborative Organization flourishing in recent years. on February 18, 2018
- Bhattacharya, I, Sharma, K (2007). India in the knowledge economy
- https://opensource.org/history
- https://www.howtogeek.com/129967/htg-explains-what-is-open-source-software-and-why-you-should-care/
- https://www.synopsys.com/glossary/what-is-open-source-software.html
- https://dwheeler.com/oss_fs_eval.html

❖❖❖❖

Unit -3

SOCIAL IMPACT

Unit Structure

3.0 OPEN SOURCE VS. CLOSED SOURCE

Open Source Software:

Open source software refers to the computer software which source is open means the general public can access and use. In short it is referred as OSS. The source code of open source software is public. It uses the code freely available on the Internet. This code can be modified by other users and organizations means that the source code is available for anyone to look at. As the software is open to the public, the result is that it constantly updates, improves and expands as more people can work on its improvement. The price of open source software is very less and there is no so much restrictions on users based on usability and modification of software.

Some examples of open source software are Fire fox, Open Office, Gimp, Alfresco, Android, Zimbra, Thunderbird, My SQL, Mailman, Moodle, TeX, Samba, Perl, PHP, KDE etc.

Closed Source Software:

Closed source software refers to the computer software which source code is closes means public is not given access to the source code. In short it is referred as CSS. In closed source software the source code is protected. Only the original authors of software can access, copy, and alter that software. In a case with closed source software, you are not purchasing the software, but only pay to use it. The price of closed source software is high and users need to have valid and authenticated license to use the software. As is issues an authenticated license so it also put a lot restriction on users based on usability and modification of software.

Some examples of closed source software are Skype, Google earth, Java, Adobe Flash, Virtual Box, Adobe Reader, Microsoft office, Microsoft Windows, WinRAR, mac OS, Adobe Flash Player etc.

Sr. No.	Open Source Software (OSS)	Closed Source Software (CSS)
01.	Open source software refers to the computer software which source is open means the general public can access and use.	Closed source software refers to the computer software which source code is closes means public is not given access to the source code.
02.	Open Source Software in short also referred as OSS.	Closed Source Software in short also referred as CSS.

03.	The source code of open source software is public.	In closed source software the source code is protected.
04.	This code can be modified by other users and organizations means that the source code is available for anyone to look at.	The only individual or organization who has created the software can only modify the code.
05.	The price of open source software is very less.	The price of closed source software is high.
06.	There is no so much restrictions on users based on usability and modification of software.	There is so much restrictions on users based on usability and modification of software.
07.	Programmers compete with each other for recognition.	Programmers do not compete with each other for recognition.
08.	Programmers freely provide improvement for recognition if their improvement is accepted.	Programmers are hired by the software firm/organization to improve the software.
09.	If the program is popular then very large number of programmers may work on the project.	There is a limitation on the number of programmers/team who will work on the project.
10.	It is purchased with its source code.	It is not purchased with its source code.
11.	Open software can be installed into any computer.	Closed software needs have a valid license before installation into any computer.
12.	Open source software fails fast and fix faster.	Closed source software has no room for failure.
13.	In closed source software no one is responsible for the software.	In closed source software the vendor is responsible if anything happened to software.
14.	Examples are Firefox, OpenOffice, Gimp, Alfresco, Android, Zimbra, Thunderbird, MySQL, Mailman, Moodle, TeX, Samba, Perl, PHP, KDE etc.	Examples are Skype, Google earth, Java, Adobe Flash, Virtual Box, Adobe Reader, Microsoft office, Microsoft Windows, WinRAR, mac OS, Adobe Flash Player etc.

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

- **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 the right fit.

- **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 modified by 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 each release. However, because of availability, the source code is open for hackers to practice 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 than compared to OSC.

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

- **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 support options are forums, useful articles, and a hired expert. However, it is not surprising that using such kind of service you will not receive a high level of response.

- **Source Code Availability**

  Open source software provides an ability to change the source code without any restrictions. 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 the source code cannot be changed or viewed. However, such limitation is what may contribute to CSS security and reliability.

- **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 to conform 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, whereas for 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, you can embrace it without being charged more for increased sales volume.

Closed source software is easier to work with for beginners or those who don't know how to code. Also, closed source websites are easier and faster to set up out of the box.

3.1 OPEN SOURCE GOVERNMENT

The concept of open government preaches a government which is highly transparent and offers mechanisms for continual public scrutiny. Open government envisions increased citizen participation and collaboration in all government proceedings through the application of modern and open technologies. The origin of this concept is traced back to the Age of Enlightenment/ Reason in the 17th and 18th centuries when the seeds of rights of free speech, expression and assembly were sown in many western nations. In the 1950s and 60s, certain laws such as freedom of information were passed which intended to foster transparent, accessible and accountable government culture. These laws came to be known as 'sunshine' laws.

If the future is a walk towards an open government, open source technologies will perhaps play the most significant role. Applications of open source software extend from computer hobbyists to professional businesses but the public sector has not always been embracing open source technologies. The reason is perhaps the misconception that open source software is time-consuming and offer unsupported solutions. The following advantages of using open source software in government may allay these unfounded concerns:

I. FLEXIBILITY

Agencies can tailor open source products, adding to or modifying the code to fit their specific needs. These modifications can include security patching for critical vulnerabilities measured in hours. Similar vendor products don't offer this flexibility.

II. MATCHES COMPETITORS

Having many sets of eyes on the code means open source tools' capabilities and functionality compete exceptionally well with commercial software.

III. AFFORDABILITY

Open source software usually is available at a much lower price point than buying a commercial product. In all cases, it is free, and valuable support options are available for popular packages.

IV. PROVEN TRACK RECORD

Many open source products are proven by having accumulated strong communities of developers over a long period of time. An example is Linux, where a large group of people, including employees of established companies, watch over security and can work with government agencies to ensure the software's safe use.

V. HIGH QUALITY

Government can be assured of high quality — defined by good testing and code review — if the open source technology is supported by a solid developer community.

VI. ACCESS TO SKILLED LABOR

Using open source software may unlock a wider base of prospective IT talent to government agencies. The number of Linux specialists, for example, gives open source users greater access to skilled technology labor.

Transparency is considered the traditional hallmark of an open government, meaning that the public should have access to government-held information and be informed of government proceedings.

Agencies also can connect easily to open source software communities online. It will be obvious which communities are developing robust products, because participants in the better communities will be actively and consistently engaged. Frequency of commits, releases and mailing list traffic are all excellent indicators of community health in the open source world. Also consider the size and diversity of companies that support a product directly through financial contribution or support and indirectly through employee involvement.

3.2 OPEN SOURCE ETHICS

3.2.1 Ethical Issue with OSS

- **Lack of Motivation**

One concern with open source software is the lack of motivation to produce quality work. Because most of the OSS developers are not paid, one may claim that these developers may not be motivated to produce reliable and good quality software. The opponents of OSS claim that if the software is not reliable then it may affect society negatively and thus not help enhance technology as much as their "reliable" commercial counterparts.

However, proponents of OSS disagree with this point. They argue that having multiple sets of eyes on code, and having it open to the public results in programmers coding better. They claim that the whole culture surrounding OSS thrives on the developer's desire for recognition of work from colleagues or utilitarian approach to information and learning and thus they are at least as likely to produce quality code as their colleagues who earn more salaries through commercialized software.

- **Ownership**

Assume that a group of researchers want to use the source code of some open source software to study metrics of coding. However, this is not the use of the software that the developers intended. The question that gets raised by this concept is whether or not researchers would need to ask the developers for permission to study the code. If they do not need the permission, then the developers might be discontented with this use and thus might be discouraged from creating new works and Open Source Software in the future. On the other hand, if they would need the permission, then it leads to even bigger problems; because of the nature of OSS there might be hundreds of developers who have worked on the product. This raises the question of "how far do the researchers need to go in order to gain permission?" Should they be required to ask every single person who has contributed to the issue for his/her consent? Should they only ask the initial core developer team? Where do they draw the line between who they should be required to ask permission from and who they do not need to? These are some of the questions that are raised with OSS development with regards to ownership and permissions.

Furthermore, if researchers do study the code, they can then evaluate which are the best and worst coders as it is easy to tell which developer has contributed to which parts of the software through version control. As the code is public information, it can be argued that publishing this information would also be legitimate. This, however seems to be an intrusion of privacy. Even if coders have remained anonymous, it is still publicly known how well or badly they code.

- **Licences and Enforceability**

Another issue related to OSS is whether or not obtaining and following through with copyright, trademark and licenses associated with the OSS product has negative affects on the software itself as well as the society as a whole.

As is evident in the previous ethical issues, some regulations are necessary to protect all parties in open source software and allow it to function with minimal disputes. However, these licences are themselves an ethical issue in terms of their enforceability and validity. Open source, however, is meant to be open and so it is a strange situation to have licences in place to regulate it. So, to what extend should the OSS licenses be treated on the level of copyright infringement and to what extend should they be taken a little more lightly due to the nature of open-source software that encourages social collaboration and the common good?

Open source software is the most ethical option for two reasons:

1. The rapid rate at which general purpose computers have and will become integrated in all aspects of our everyday life.
2. The benefit to the wider community that comes with knowledge sharing and collaboration.

Open source software is the most ethical option because of the ways computers have and will become integrated in all aspects of our life. Without access to the source code, we put our privacy, lives, and the lives of others in danger. Moreover, our lives and quality of life also depends on our ability to choose what software runs and is trusted on the devices we own.

When we expose and make free the code that runs on our computers, we invite others to learn from our knowledge and share alike, with the added benefit that we often get our software updates faster and with greater regularity. Just as no one person can spot all of the bugs in a program, no one person can create the most efficient program. Open source software allows us to select from a variety of diverse solutions that ultimately improve our quality of life, and provides us with the ability to modify that software to suit our needs. In a closed source world, we depend on the manufacturer to make all of the right decisions and to anticipate all of the right design and security questions, which is a highly unreasonable request.

The value of ethical OSS licenses and next steps support for so-called ethical OSS licenses is not universal amongst the OSS community and it not surprising that the recent release of the Hippocratic License has engendered its fair share of bricks and bouquets. Given accusations by Emke that the OSI has prioritized software freedom over ethical concerns, it was predictable that the OSI would essentially assert that the Hippocratic License is not certifiable as a legitimate OSS license (presumably because it fails to pass two critical Open Source Definition requirements, namely numbers 5 (No Discrimination Against Persons or Groups) and 6 (No Discrimination Against Fields of Endeavor). Many ethical licenses may fail on these grounds, particularly the requirement that an OSS license must not restrict anyone from making use of the program in a specific field of endeavor, such as a particular business, or from being used genetic research by way of example.

Critics have also been quick to point out the challenges of ethical OSS licenses: enforceability, for one (at worst a license can be “revoked,” but it may be difficult to force anyone to actually change their behaviour or stop using the code), and practicality (who gets to decide what is an “adversely addictive behaviour”?). At the same time, ethical OSS pundits such as Chris Jenson have argued that the GPL was and is also rooted in a strong ideological viewpoint (software should be free, as in speech) and while such views are now widely accepted, “when it first appeared it was a very new and strange idea about licensing software, and was met with a lot of resistance.”

Moreover, many would assert (myself included) that the real value of ethical OSS licenses is that they are inherently disruptive by bringing attention to these ethical issues, challenging the status quo (including the Open Source Definition), and spreading the idea (in Jenson’s words) that software “should be used for the betterment of the world” and that “as developers we can take responsibility for how our code is used.” It ultimately may not matter if such licenses gain “official” certification, and the approval of organizations such as the OSI if they otherwise have an impact.

Arguably the more narrowly drafted ethical OSS licenses, such as the Anti-996 License, stand a better chance of being voluntarily adopted (in comparison with the sprawling Just World License); and the fact that so many code projects and companies appear to have adopted the Anti-996 License to date is impressive.

Notwithstanding its initial protests, even the OSI may eventually be forced to reconsider and update its OSI Definition, given that the document dates back to the 1990s (an eternity in tech years). In an age of increased technology worker activism, where U.S. employees have successfully pressured their employers to cease certain kinds of work for the U.S. military (e.g., Google's aerial drone imagery analysis), ICE and U.S. Customs and Border Protection, it may be difficult to fully ignore the potential impact of ethical OSS.

Only time will tell whether certain of the ethical OSS licenses will succeed or fall into obscurity.

3.3 SOCIAL AND FINANCIAL IMPACTS OF OPEN SOURCE TECHNOLOGY

Today, mass collaboration is changing the essential structure of organizations and reshaping how these companies work in our exceptionally serious condition. The collaborative effort, energized by open philosophies and companion generation, is constraining the administration to reconsider their methodologies. Collaborations that have emerged are breaking the obstructions and making open spaces where all can develop and add to push forward the limits of their organizations just as the limits of ventures, they work in the incomparable British economics specialist, makes a solid contention that one reason for the structure of vertically incorporated collaborations is the "cost of the exchange." Perform an exchange inside your firm in particular on the off chance that it is less expensive than performing it remotely or in the commercial center. The Internet blast and the advancement of open-source software and pooled frameworks have made it feasible for online organizations to keep these exchange costs low. Wear Tapscott, the creator of Wikinomics, analyzes this thought: "Exchange costs despite everything exist, except now they're regularly graver in companies than in the commercial center."
Despite the considerable number of advantages (as far as quality, speed, and riches) that open source and the shared method of undertaking ventures have produced, there is still some misconception and hole in the valuation for these huge changes. An organization despite everything confines their appreciation of open source as free programming that sucks up the abundance of a solid entrepreneur society. They consider liberated to be as a risk to the venture however miss the multi-billion-dollar biological system that it has made from which organizations everything being equal and types are profiting.

Collaborative Organization flourishing in recent years

The computerized upheaval, additionally called the third unrest, has changed the whole scene of the business world. After the mechanical transformation, no other unrest has changed the texture of the general public as the Internet has transformed it. It has offered to ascend to collaborations that flourish with volunteers, peer creation, and joint effort. Wikipedia, Word Press, Red Hat, the Mozilla Foundation and a lot more are contending today with probably the best-financed and creative endeavors over the globe.

The parameters of this challenge are represented by cost as well as characterized by quality too. A Wikimedia traffic investigation report in 2012 shows that Google Chrome has a bigger market than Internet Explorer and browser Mozilla Firefox has a noteworthy piece of the pie. In like manner, Red Hat Enterprise Linux has generally actualized in practically all the enormous money related organizations due to the expense as well as a result of the solidness it adds to the unpredictable innovation frameworks in budget confined organizations.

Indeed, even a portion of the collaborations that have a background marked by restricting and harpooning open source advancements are presently opening up to coordinated efforts to make win-win circumstances. Microsoft is the greatest model. It's entirely claimed backup, Microsoft open advances gathering, follows a network-driven way to deal with make inventive arrangements. One late declaration was the dispatch of VM Depot, a community is driven inventory of open-source virtual machine pictures for Windows Azure.

People starting up their own businesses

The Internet is perhaps the best thing that has happened to mankind. In addition to the fact that it opens the world up to an individual (and the other way around), it has become a core of worldwide monetary action. An ever-increasing number of individuals today are making their living by selling bits and bytes. The expense of beginning an online business is very low compared with beginning a business that depends on physical channels. This minimal effort of bootstrapping a business joined with the imaginative idea of the Internet has urged millions to start their endeavors. The ease of beginning an online business has become easy financially due to the accessibility of open source software and foundation. The free LAMP programming stack, which establishes Linux, Apache, My SQL, and PHP, has made it feasible for imaginative and keen individuals with thoughts to begin organizations on the Internet that are assuming a positive job in pushing forward mankind.

The open-source development and its procedures have contributed altogether to the business world and made environments that have emphatically affected all ventures and billions of individuals over the globe. Also, this development has generally been powered by a huge number of volunteers who add to these activities for a wide scope of reasons, including to develop their systems, improve their resumes, refine their abilities, and only for doing social great.

In the expressions of Jimmy Wales, the author of Wikipedia: "We are assembling to manufacture this asset that will be made accessible to every one of the individuals of the world for nothing. That is an objective that individuals can get behind".

3.4 SHARED SOFTWARE

Shared Software means all Software owned or licensed by the Seller and the Seller Subsidiaries as of the Closing Date that is necessary in order to conduct the Business substantially in the manner and to the extent currently conducted or used by Seller in connection with the Business as of the Closing Date (other than the Transferred Software), where the functionality of such Software is material to the Business as of the Closing Date and cannot be replaced with Software providing comparable levels of performance and functionality at a cost of less than $250,000

Any Party shall have the right to bring an Action for infringement, misappropriation, or other violation with respect to the Shared Software ("Enforcement Action") without the consent of the other Party, except that the Parties may cooperate, at their respective own expense, in any Enforcement

Action with respect to alleged infringement, misappropriation, or other violation of Shared Software.

At any time upon the disclosing Party's written request, the receiving Party shall promptly return to the other Party, or destroy, all Confidential Information of the disclosing Party obtained by the receiving Party under the Agreement, and all copies and reproductions thereof, except for Confidential Information related to or associated with the Shared Software, including but not limited to the source code for the Shared Software.

However, the Parties shall not file for any intellectual property protection worldwide for any Shared Software or Software Improvements without written permission from the other Party obtained in advance of such filing.
Subject to the terms of this Agreement, each Party may use and commercially exploit the Shared Software for its own benefit in any manner without the consent of the other Party and without any obligation, accounting, or payment of any fee to the other Party.

3.5 SHARED SOURCE

Shared source is a software licensing concept that is more open than the proprietary approach to licensing but more restricted than the open source model. Under a shared source program license, authorized parties are granted full or partial access to source code. Typically, those granted access can view source code but cannot alter it for any commercial purpose. Some shared source programs allow only viewing of code; others allow non-commercial alteration and/or redistribution. Microsoft originated the shared source approach, which has since been adopted by other major industry players, including Hewlett-Packard and Sun Microsystems.

The shared source model offers fewer benefits than the open source model. It lacks, for example, the collaborative improvement process promoted by the open source approach. However, even the ability to view source code can be helpful. Shared source code can help developers ensure compatibility with existing programs and can make it easier to review source code for security purposes.

Critics have described shared source as a marketing ploy and suggested that the approach could pose a threat to the purity of the open source model. In a paper called "Shared Source: A Dangerous Virus," the Open Source Initiative called Microsoft's shared source program "a trap for the unwary" and warned that developers who'd been exposed to it should be considered "contaminated" and not assigned to projects that were competitive with Microsoft products.

3.6 OPEN SOURCE IN GOVERNMENT

Open source can be seen as a technical expression of democratic government as open source is a result of public accessibility, open exchange and collaborative participation. It thrives on transparency, meritocracy and community development. The goal of open source and democracy is loosely the same – ensure more control and create a better future.

Open source software has so many potential benefits that in many cases, agencies should at least consider it when technology is a factor in solving problems or expanding services. And because open source software is so pervasive, all major products are graded just as commercial products are. To find open source software reviews, government agencies can access information technology research conducted by organizations such as Forrester and Gartner. These independent firms do deep dives to analyze technology products, including open source software.

Five open source software products that can aid government:

For agencies that want to consider open source software, five tools in particular offer significant potential advantages to government:

1. **Open Stack** is a set of tools that allows users to create, automate and manage both public and private clouds at minimal expense. This technology, originally developed by NASA, is a good example of government developing open source software to meet a particular agency need, and then putting the software back into the open source community so others can add to and benefit from it. With Open Stack, government can set up its own cloud to hold data it doesn't want in a public cloud.

2. **Jenkins** is a continuous delivery tool that builds and tests software after every change. Once development and testing are complete, Jenkins can deploy new code to production with the push of a button, so there's no down time in making the upgraded product available to constituents.

3. **Docker** 1.0 was released in mid-2014, and the buzz around it has yet to die down. This tool "containerizes" applications with everything they need to run, allowing them to be moved around the cloud and ensuring they will run as well in one technology environment as in another. Like putting cargo into a container makes it easier to move from place to place, Docker streamlines the process of moving from a legacy server to the cloud or moving from one cloud server to another. This reduces government costs by preventing agencies from being held captive by an existing cloud provider's prices.

4. **Spacewalk** automates hardware and software inventorying, and software installations and updates for Linux server environments. By using Spacewalk and Linux together, an agency can maintain a low-risk security posture by deploying monthly patches at the touch of a button.

5. **Drupal**, an open source content management system, is used to build websites and author content. Used by 37 percent of .gov websites, Drupal was the CMS technology of choice for the state of Colorado, whose proprietary tool had become inflexible and unstable after reaching more than 100 customers on the platform. The legacy solution also had been purchased by a large company, and its price was expected to increase significantly. After the state conducted an analysis of alternatives, Drupal stood out. With more than 1 million users worldwide, it offers ample online resources and contractors available for support, provides a relatively simple interface, is adequately flexible to support growth without significant investment and reduces the required implementation time by nearly 15 times. The effort has been wildly successful, and the state government portal has launched three distributions of Drupal to 174 state agencies in Colorado in the past two years.

Like any technology product, open source software isn't right for every application in every government agency. But its capability for transforming the delivery of digital government services means there's a best-use place for open source software in every government technology portfolio.

3.7 SUMMARY

This Chapter analyses the impact of public policies supporting open source software (OSS). Users can be divided between those who know about the existence of OSS, the "informed" adopters, and the "uninformed" ones; the presence of uniformed users yields to market failures that justify government intervention. We study three policies: i) mandatory adoption, when government forces public agencies, schools and universities to adopt OSS, ii) information campaign, when the government informs the uninformed users about the existence and the characteristics of OSS and, iii) subsidisation, when consumers are payed a subsidy when adopting OSS. We show that the second policy enhances welfare, the third is always welfare decreasing while mandatory adoption can be either good or bad for society depending on the number of informed and uninformed adopters. We extend the model to the presence of network effects and we show that strong externalities require "drastic" policies.

3.8 REFERENCES

- Monika Sharma, learn about the social and financial impacts of Open Source, collaborative Organization flourishing in recent years. on February 18, 2018
- Bhattacharya, I, Sharma, K (2007). India in the knowledge economy
- Gera, S., R. Roy, and T. Songsakul (2006)
- Wheeler, David A. (May 8, 2014). "Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)
- https://www.includehelp.com
- Golden, B.: Open Source Maturity Model © Navica, http://www.navicasoft.com/ pages/osmmoverview.htm
- Method for Qualification and Selection of Open Source software (QSOS) version 1.6 © Atos Origin (April 2006), http://qsos.org/
- https://dwheeler.com/oss_fs_eval.html

❖❖❖❖

UNIT -4

CASE STUDY ON LINUX TECHNOLOGY AND OPEN SOURCE SOFTWARES

Unit Structure

4.0 LINUX AND THE GNU SYSTEM:

Many computer users run a modified version of the GNU system every day, without realizing it. Through a peculiar turn of events, the version of GNU which is widely used today is often called “Linux”, and many of its users are not aware that it is basically the GNU system, developed by the GNU Project.

There really is a Linux, and these people are using it, but it is just a part of the system they use. Linux is the kernel: the program in the system that allocates the machine's resources to the other programs that you run. The kernel is an essential part of an operating system, but useless by itself; it can only function in the context of a complete operating system. Linux is normally used in combination with the GNU operating system: the whole system is basically GNU with Linux added, or GNU/Linux. All the so-called “Linux” distributions are really distributions of GNU/Linux.

Many users do not understand the difference between the kernel, which is Linux, and the whole system, which they also call “Linux”. The ambiguous use of the name doesn't help people understand. These users often think that Linus Torvalds developed the whole operating system in 1991, with a bit of help.

Programmers generally know that Linux is a kernel. But since they have generally heard the whole system called “Linux” as well, they often envisage a history that would justify naming the whole system after the kernel. For example, many believe that once Linus Torvalds finished writing Linux, the kernel, its users looked around for other free software to go with it, and found that (for no particular reason) most everything necessary to make a Unix-like system was already available.

What they found was no accident—it was the not-quite-complete GNU system. The available free software added up to a complete system because the GNU Project had been working since 1984 to make one. In the GNU Manifesto we set forth the goal of developing a free Unix-like system, called GNU. The Initial Announcement of the GNU Project also outlines some of the original plans for the GNU system. By the time Linux was started, GNU was almost finished.

Most free software projects have the goal of developing a particular program for a particular job. For example, Linus Torvalds set out to write a Unix-like kernel (Linux); Donald Knuth set out to write a text formatter (TeX); Bob Scheifler set out to develop a window system (the X Window System). It's natural to measure the contribution of this kind of project by specific programs that came from the project.

If we tried to measure the GNU Project's contribution in this way, what would we conclude? One CD-ROM vendor found that in their “Linux distribution”, GNU software was the largest single contingent, around 28% of the total source code, and this included some of the essential major components without which there could be no system. Linux itself was about 3%. (The proportions in 2008 are similar: in the “main” repository of gNewSense, Linux is 1.5% and GNU packages are 15%.) So if you were going to pick a name for the system based on who wrote the programs in the system, the most appropriate single choice would be “GNU”.

But that is not the deepest way to consider the question. The GNU Project was not, is not, a project to develop specific software packages. It was not a project to develop a C compiler, although we did that. It was not a project to develop a text editor, although we developed one. The GNU Project set out to develop a complete free Unix-like system: GNU.

Many people have made major contributions to the free software in the system, and they all deserve credit for their software. But the reason it is an integrated system—and not just a collection of useful programs—is because the GNU Project set out to make it one. We made a list of the programs needed to make a complete free system, and we systematically found, wrote, or found people to write everything on the list. We wrote essential but unexciting (1) components because you can't have a system without them. Some of our system components, the programming tools, became popular on their own among programmers, but we wrote many components that are not tools (2). We even developed a chess game, GNU Chess, because a complete system needs games too.

By the early 90s we had put together the whole system aside from the kernel. We had also started a kernel, the GNU Hurd, which runs on top of Mach. Developing this kernel has been a lot harder than we expected; the GNU Hurd started working reliably in 2001, but it is a long way from being ready for people to use in general.

Fortunately, we didn't have to wait for the Hurd, because of Linux. Once Torvalds freed Linux in 1992, it fit into the last major gap in the GNU system. People could then combine Linux with the GNU system to make a complete free system — a version of the GNU system which also contained Linux. The GNU/Linux system, in other words.

Making them work well together was not a trivial job. Some GNU components(3) needed substantial change to work with Linux. Integrating a complete system as a distribution that would work “out of the box” was a big job, too. It required addressing the issue of how to install and boot the system—a problem we had not tackled, because we hadn't yet reached that point. Thus, the people who developed the various system distributions did a lot of essential work. But it was work that, in the nature of things, was surely going to be done by someone.

The GNU Project supports GNU/Linux systems as well as the GNU system. The FSF funded the rewriting of the Linux-related extensions to the GNU C library, so that now they are well integrated, and the newest GNU/Linux systems use the current library release with no changes. The FSF also funded an early stage of the development of Debian GNU/Linux.

Today there are many different variants of the GNU/Linux system (often called “distros”). Most of them include nonfree programs—their developers follow the “open source” philosophy associated with Linux rather than the “free software” philosophy of GNU. But there are also completely free GNU/Linux distros. The FSF supports computer facilities for a few of them.

Making a free GNU/Linux distribution is not just a matter of eliminating various nonfree programs. Nowadays, the usual version of Linux contains nonfree programs too. These programs are intended to be loaded into I/O devices when the system starts, and they are included, as long series of numbers, in the "source code" of Linux. Thus, maintaining free GNU/Linux distributions now entails maintaining a free version of Linux too.

Whether you use GNU/Linux or not, please don't confuse the public by using the name “Linux” ambiguously. Linux is the kernel, one of the essential major components of the system. The system as a whole is basically the GNU system, with Linux added. When you're talking about this combination, please call it “GNU/Linux”.

4.1 CASE STUDY ON ANDROID

1. What is the Android operating system?

The Android OS was originally created by Android, Inc., which was bought by Google in 2005. Google teamed up with other companies to form the Open Handset Alliance (OHA), which has become responsible for the continued development of the Android OS.

Android’s underlying kernel is based on Linux, but it has been customized to suit Google’s directions. There is no support for the GNU libraries and it does not have a native X Windows system. Inside the Linux kernel are found drivers for the display, camera, flash memory, keypad, WiFi and audio. The Linux kernel serves as an abstraction between the hardware and the rest of the software on the phone. It also takes care of core system services like security, memory management, process management and the network stack.

The Android OS is designed for phones. Its many features include:

- Integrated browser, based on the open source WebKit engine
- Optimized 2D and 3D graphics, multimedia and GSM connectivity
- Bluetooth
- EDGE
- 3G
- WiFi SQ Lite Camera GPS Compass Accelerometer

Software developers who want to create applications for the Android OS can download the Android Software Development Kit (SDK) for a specific version. The SDK includes a debugger, libraries, an emulator, some documentation, sample code and tutorials. For faster development, interested parties can use graphical integrated development environments (IDEs) such as Eclipse to write applications in Java.

Android story

- Android Inc was founded in Palo Alto, California, United States by Andy Rubin, Rich Miner, Nick Sears & Chris White -- Oct 2003
- Google acquired Android Inc -Aug2005
- The Open Handset Alliance, a consortium of several companies was formed -5 thNov2007
- Android beta SDK released -12thNov2007

Android Versions:
Android Version 2.x.x – Gingerbread (6th December, 2010)

The Android Version Gingerbread brought a revolution into the world of mobile communication. Since the release of the first Android Version, Android had been trying to make its way to the core of the mobility market but their journey actually started with the release of this Android Version

- Google added an intelligent User Interface into this particular Android Version
- New and improved keyboard the for the ease of the uses
- Added the feature of copy/paste
- Power Efficiency for the efficient use of the mobile battery
- Social Network related features added
- NFC or Near Field Communication Support added
- Video Call Support

Besides, the above features that were added onto the Android Version there were also the ones that were added for the ease of developers. Basically, all the necessary things that a developer would require for developing anything and everything related to the android
platforms were added into this Android Version.

Android Version 3.x.x – Honeycomb (22nd February, 2011)

At the time of the release of this Android Version, tablets devices were getting famous in the market, so Android rather took a risky turn which did not go in their favor. Honeycomb was basically for the purpose of enriching the tablet UI. The list of features added are as below

- Multi Core Support to improve processing
- Tablet Support
- 3D UI Updated
    - Customizable home screens
    - Recent applications view
- New Keyboard layout
- Media Transfer Protocol
- Google Talk video Chat

- Private Browsing for privacy improvement
- HTTP live streaming

A version update of this Android Version was released in the year 2012 that brought the feature of “Pay as you go” but since the Android Version was not as famous as it should have been, it did not go viral.

Android Version 4.0.x – Ice Cream Sandwich (18th October, 2011)

After the not so popular Android Version, codenamed Honeycomb, came the Ice Cream Sandwich which continued with the popularity of the Android Operating System. It came with several bug fixes and a large list of features were also added into the Android Operating System. The Features are as follows:

- New Lock Screen Actions
- Improved text input and spell-checking
- Control of the Network data
- Email app support
- Wi-Fi direct
- Bluetooth health device profile
- Social Stream to keep the contacts updated
- Video Stabilization
- QVGA video resolution API access
- Calendar provider updates
- Smoother screen rotation

Android Version 4.1.x – 4.3.x – Jelly Bean (9th July, 2012)

By the time of the release of Android Versions 4.0.x, codenamed Ice Cream Sandwich. Android had pulled away most of the users from the competitors and Google totally knew about what to change in their next Android Version. This step was a crucial one because either people would move away from Android or stick to it for as long as it exists.

- Google Now
- Voice Search
- Speed Enhancements
- Camera app improvements
- External Keyboards and Gesture mode – improving accessibility
- Lock screen widgets
- 4K resolution support
- Restricted profiles for tablets
- Dial Pad auto-complete
- Shows the percentage of download and the time remaining.

Although, the Code Name was changed from Ice Cream Sandwich to Jelly Bean but the numeric series had been kept the same as the previous one. And to be practical there was not much of a difference in the feel of this version.

Android Version 4.4.x – Kitkat (31st October, 2013)

This was the most controversial Android Version of all time. The ever awaited LG Google Nexus 5 and the Android Version, Codenamed Kitkat, were to make their way into the spotlight together. Google had gone way too late on their release of this version and their smartphone. Although, both of them were worth waiting for. The features included with in this version of Android were as follows:

- Screen Recording
- Translucent System UI
- Better and Enhanced notification access
- Performance improvements

Although, the list of changes might seem shorter than expected but Google actually made it look like something huge and they did succeed in the process. The contradictory part is also true, since a lot of people choose other devices over the Nexus because they arrived earlier and they seemed to be a better buy.

Android Version 5.0 – Lollipop (17th October, 2014)

With the release of this Android version it seemed like Kitkat (The previous Android Version) had not been in the market for long enough to make its way towards the hearts of the Android users. This version came with the Motorola Nexus 6 from the LG Nexus, which actually made it seem like they must have made more changes and after using this Android Version you would actually feel like that is true but that would only be true when it comes to the UI and the keyboard features.

Although this Android Version is still in its immature stage but the features they have come up with till date are as follows:

- New Design – Material Design
- Speed Improvement
- Battery Efficiency

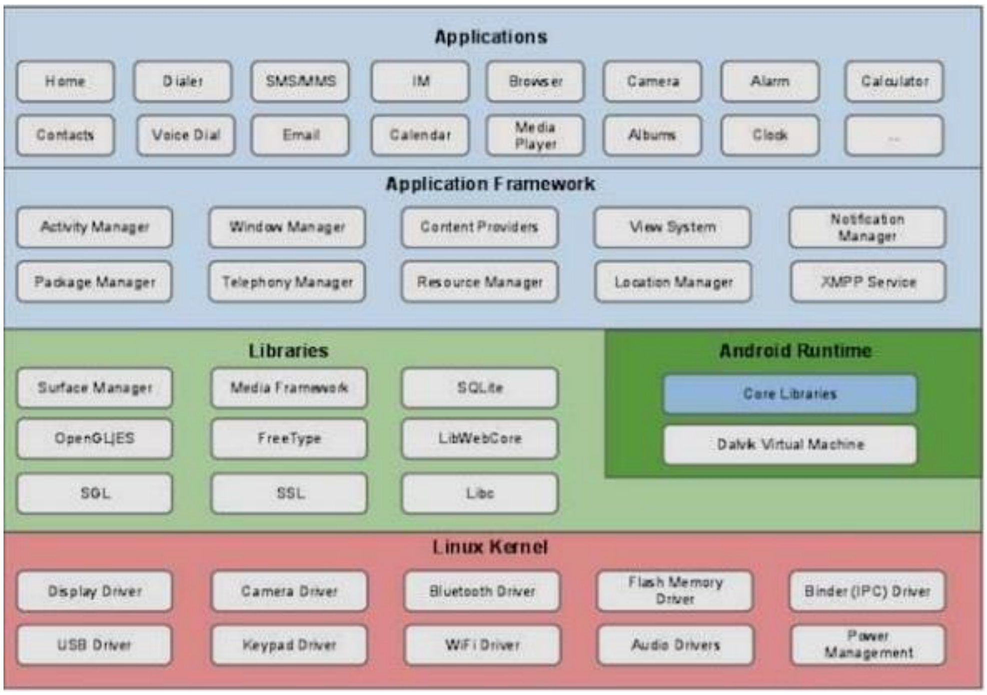
Android 6.0 – Marshmallow May 28, 2015

- Android 6.0 "Marshmallow" was unveiled under the codename "Android M" during Google I/O on May 28, 2015, for the Nexus 5 and Nexus 6 phones, Nexus 9 tablet, and Nexus Player set-top box, under the build number MPZ44Q.

- Requirements for the minimum amount of RAM for devices running Android 5.1 range from 512 MB of RAM for normal-density screens, to about 1.8 GB for high-density screens. The recommendation for Android 4.4 is to have at least 512 MB of RAM,[213] while for "low RAM" devices 340 MB is the required minimum amount that does not include memory dedicated to various hardware components such as the base band processor.

- The Android N Developer Preview is already here and this will be followed by monthly updates until the final version. That final version will likely come around Nexus time – late September or early October – with Android N availability for other manufacturers and devices in the six or so months to follow.

Android Architecture with diagram.



Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like the camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database

4.3 CASE STUDY ON MOZILLA FIREFOX

Mozilla Firefox is a Free/Libre/Open Source (FLOSS) browser supported by the Mozilla Foundation. This browser was recently released and has met with considerable success- it has been downloaded more than 20 million times and has already taken considerable market share from its prime competitor- Microsoft's

Internet Explorer.

Background:

The Mozilla project is an offshoot of the Netscape browser. Readers are encouraged to see the timeline at http://www. holgermetzger. de/Netscape_History.html for a detailed set of events relating to Netscape. A longer description of the first round of the browser wars is available in Cusmano and Yoffie (1998). In short, Mozilla was released as an open source version of Netscape in January 1998. Since that time, Mozilla has released many versions of its browsers. Netscape and AOL use its browsers. Firefox is the latest version of the Mozilla browser.

A small team of three motivated and talented individuals, Blake Ross, AsaDotzler and Ben Goodger, developed Mozilla Firefox

Compatibility with other operating systems (Linux, Windows and Apple)

System overview:

Mozilla and Firefox consist of about two thousand C and C++ source code files (".c", ".cp" and" cpp" files) and about 0.8 to 0.9 million uncommented lines of code in each release.

Figure 5(a) counts Mozilla and Firefox releases by ULOC, while Figure 5(b) counts NOF, the number of files. A file is considered to be shared if and only if two versions of the file in the sibling releases have identical ULOC. It is shown that the overall size of Mozilla and Firefox releases are relatively stable in terms of NOF and ULOC during the one year and a half period of time. However, the percentage of shared code changes significantly across some of the releases.

Conditions That Facilitated the Success Of Firefox:

Complacent Competition: At the time of Mozilla Firefox' launch, the largest competitor, Microsoft's Internet Explorer, had become a static product. Microsoft had made a strategic decision to link IE to its operating system. What this meant is that newer versions of IE would only be available on newer versions of Windows. As a result, the only changes to the product were related to the security vulnerabilities of the product. This lack of product innovation left the door open for competitors such as Mozilla Firefox. Microsoft's decision to bundle the innovation of IE with that of the Windows operating system may prove to have been a major strategic error. This was especially so since the company is involved in a major overhauling of their Windows operating system as part of the Longhorn project.

Product Superiority: Many impartial observers agree with Blake and have concluded that Mozilla Firefox is asuperior alternative to its competitors. Here, I will argue that Firefox offers three newideas- compatibility, tabbing and better security. Readers are referred to an older
document
http://web.archive.org/web/20040210101506/http://www.mozilla.org/products/firefox/why/) authored by Ben Goodger for a longer list of features.

Compatibility with other operating systems (Linux, Windows and Apple): Internet Explorer is compatible only with Windows-based operating systems(specifically Microsoft Windows® 98, Windows 2000, or Windows XP).

In contrast, Firefox is compatible with Linux, Windows and Apple operating systems. This widens the potential audience for the product.

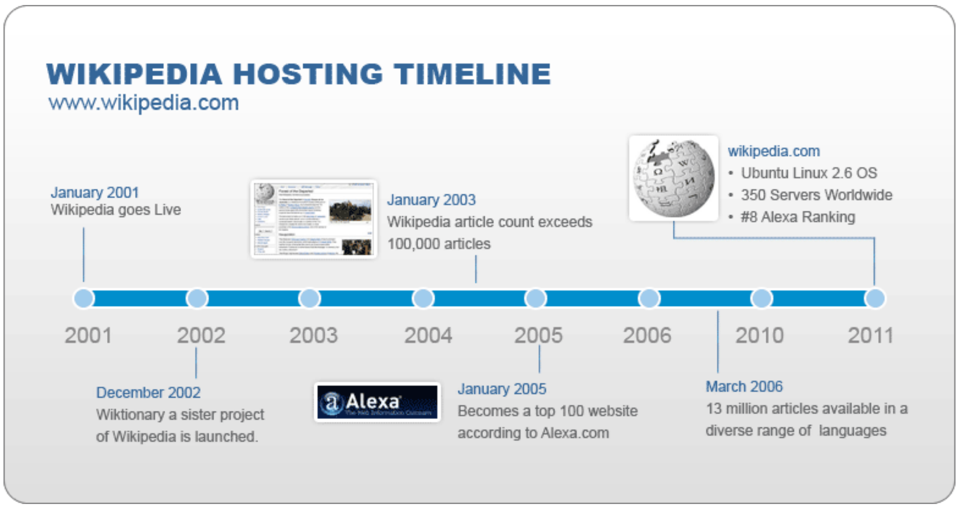
Security: The use of open source as the development methodology is a sound way to enhance the security of the product. Open source 18products allow anybody to inspect the code base, thus enhancing the chances that vulnerabilities and bugs would be detected. There are some indications that Mozilla Firefox may have its own security flaws, however.

Tabbing: Most browsers open a new link in a new window. Mozilla Firefox introduced a new feature called tabbing. This allows the user to open multiple pages in one window.

Presence of Marketing Leader: The marketing leader behind Firefox' campaign is Rob Davis, a marketing professional with experience in political campaigns (see his web site
http://www.playpolitics.org). Rob contacted the Spread Firefox team after his computer had been infected by a virus and volunteered his time to run the campaign. His main focus was on the New York Times advertising campaign.

Volunteer Support: The credit for the success of Firefox must mostly be given to the volunteers for all their hard work. Since Firefox had released a preview version before its official release, it was able to ascertain the level of interest in the community. This gave the team considerable confidence when soliciting funds for the New York Times ad. Volunteers participated in many activities on the site.

4.4 CASE STUDY ON WIKIPEDIA


WIKIPEDIA HOSTING TIMELINE
www.wikipedia.com
January 2001
Wikipedia goes Live
January 2003
Wikipedia article count exceeds 100,000 articles
wikipedia.com
• Ubuntu Linux 2.6 OS
• 350 Servers Worldwide
• #8 Alexa Ranking
2001
2002
2003
2004
2005
2006
2010
2011
December 2002
Wiktionary a sister project of Wikipedia is launched.
Alexa
January 2005
Becomes a top 100 website according to Alexa.com
March 2006
13 million articles available in a diverse range of languages

Wikipedia has more than thirteen million articles in several different languages.

System Specifications:

- **Bandwidth (Why Low?):** Wikipedia's processes are extremely simple and basic. While receiving an enormous amount of constant traffic, Wikipedia only has to send a fixed page of text and the occasional image.

- **CPU (Why Low?):** The encyclopedia pages of Wikipedia are very easy to power and don't require too many resources. The processing power used most likely powers the indexing, links, and search databases.
- **Disk (Why High?):** With the millions of articles available to the public in dozens of languages, Wikipedia has to have a large amount of disk space in order to store them all.
- **RAM (Why Low?):** The individual encyclopedia pages are not heavy on content and do not require a large amount of RAM to be displayed.
- **Scalability (Why High?):** With a constant flow of traffic and having become a service in which people rely on, Wikipedia has developed a very scalable and efficient server solution in order to meet the demands of its website.

Overview:

Wikipedia, one of the first online encyclopedias, has come a long way since its beginnings in 2001. Now, just after its ten-year anniversary, Wikipedia has more than thirteen million articles and over one hundred and sixty different language editions and is currently ranked number 8 on alexa.com with more than 10% of the world's Internet users visiting the site.

History:

In January 2001, Wikipedia went live. Originally, Wikipedia was only formed as a Wiki to help in the Nupedia project. Larry Sanger, Wikipedia's chief instigator, said it originally was an "idea to add a little feature to Nupedia." In only two months, Wikipedia had more than a thousand articles, and by October 2001, that number rose to thirteen thousand.

In December of 2002, Wikipedia's sister project, Wiktionary, was launched. By January 2003, Wikimedia's article count exceeded 100,000 articles. In June of the same year, Jimmy Wales, another founder of Wikipedia, formed Wikimedia, which now manages Wikipedia and several other sister projects.

In the following January, Wikipedia was shut down for a week after a significant computer crash. This prompted Wales to start a fundraiser to buy more computers. The nine computers he bought were then relocated to Florida. By April of that year, Wikipedia had a quarter of a million articles.

In 2004 and 2005, Wikipedia won many awards and received a significant amount of recognition. In 2005, it became a top 100 website according to Alexa.com. Wikipedia's one millionth article was published in March of 2006 and in just three years that number would grow to a total of thirteen million articles available in a diverse range of languages.

Features:

Wikipedia is a free, non-profit online encyclopedia that has surpassed more than 3.5 million articles in English alone and is one of the most highly trafficked sites on the web. The articles on Wikipedia are written and edited by thousands of volunteer contributors as it is funded solely by donations. Wikipedia founders vowed never to have advertisements on its site, which is a promise they have diligently kept.

Wikipedia features a simple search engine and a consistent format.

System:

Wikipedia uses MediaWiki software to run its website. Its main U.S. data center, consisting of about 300 servers, is located in Tampa, Florida. Wikipedia also has a European data center in Amsterdam, EvoSwitch, where they have approximately 50 servers. There are also tentative plans to secure another data center in Virginia. Wikipedia currently uses Ubuntu Linux 2.6 as their operating system with a standard LAMP package.

Summary:

Wikipedia, a non-profit online encyclopedia, has grown from just over a thousand articles at its beginnings in 2001 to over thirteen million articles in several different languages today. Wikipedia has an Ubuntu Linux 2.6 operating system and has one U.S. data center in Florida and one international center in Amsterdam. It was originally formed as a new feature for Nupedia, but now has several sister projects, such as Wiktionary, under the head organization, Wikimedia.

4.5 CASE STUDY ON DRUPAL

How Drupal saved its users millions of hours thanks to CKEditor

Everyone knows Drupal. It's the leading CMS made by one of the largest open source communities. It's designed to build outstanding digital experiences that reach your audience across multiple channels. CKEditor empowers content creators on hundreds of thousands of Drupal 8 sites. That translates to millions of users per day typing text, linking content, uploading images and organizing the world's information.

Challenge: move Drupal to the next level

Drupal has been around since 2001. And yet even Drupal 7, shipped in 2011, did not include a rich text editor. It was one of Drupal's top pain points. In surveys, it came up time and time again that a built-in rich text editor was crucial for Drupal's future. Drupal needed a well-integrated editor that supports per-site custom needs. But the process of choosing one involved a lot of work from the community. First they needed to build a consensus on which editor to choose. And then they had to do all the integration work in Drupal core.

To make this happen, Drupal's founder started the Spark initiative, whose goal was to bring the authoring experience of Drupal 8 to the next level. It included integrating a rich text editor capable of working on the front-end, for so called in-place editing. Another crucial feature to have was excellent image upload support with alignment and captioning.

Solution: mature editor with in-place editing support

At the time, very few rich text editors were capable of in-place editing. The Drupal community initially ended up selecting a rich text editor that was not very mature but had one feature — "Aloha Blocks" — which could play an important role in the Drupal ecosystem. Over time, it became clear that that rich text editor's lack of maturity and poor accessibility was going to be a huge stumbling block.

“And that's when we found out that CKEditor 4 had been in development and was also going to have support for in-place editing!”, recalls Wim Leers, one of a few dozen Drupal core maintainers and responsible for CKEditor integration. “It had far superior maturity and accessibility, as well as a much bigger development team behind it. We got strong guarantees for clean HTML, including Advanced Content Filter. And CKEditor Widgets could easily replace Aloha Blocks. So we made the difficult decision to switch to CKEditor.”

Leers stresses the maturity and stability of the editor. “The number of times CKEditor broke something in the past five years can probably be counted on two hands. Which is very impressive, considering how many different ways it's used on hundreds of thousands of Drupal 8 sites out there!”

Result: countless hours of development saved

Before the integration just about every Drupal-based project had to go through a painful process of evaluating the different available rich text editors. Then after making a selection they had to figure out how to get it integrated. “Almost every project building a Drupal site was spending at least a few and often dozens of hours on this”, says Leers. “Multiplied by more than a million Drupal sites out there, it adds up to millions of hours!”

This is no longer the case thanks to CKEditor being enabled out-of-the-box in Drupal 8 core and its deep integration. Countless hours of development time and frustration are saved, not to mention the reduction in risk. Rich text editing is simply a solved problem since Drupal 8. Both developers and users can focus on what really matters to them: creating amazing content and building ambitious digital experiences.

4.6 CASE STUDY ON WORD PRESS

The Project — Neighborhood Event

The LoLa Art Crawl is an annual neighborhood art event with over 100 artists, dozens of business sponsors, and excellent print graphics—but they had only a small placeholder WordPress site with a few static pages and PDF files of their printed materials for download. The group wanted a better website presence, artists wanted more recognition, and sponsors wanted more exposure; but as a minimally funded volunteer group, they didn’t know what to do. Word &Image volunteered to provide pro bono development services as a donation to the local community.

The Challenge — Lack of Capability

Rather than push downloadable print content to the web, we advised the group to create an online experience that would:

- List each artist with a profile page of text and art
- Spotlight sponsors with promotional profiles and featured placements
- Work on mobile devices during the event in the field
- Provide geo mapping to sites on smart phones.
- Provide news updates and social media integration

The artists group had stable funding with a nonprofit fiscal agent and local business sponsors along with enough volunteers to manage the event, but they did not have any capabilities to contribute to a web development project. Without a dedicated organizational staff and budget, how could they add a complex website to accommodate the growing needs of artists, sponsors, and art patrons?

Why WordPress — Open Source to the Rescue

The graphic designer for the print materials considered using a low-cost commercial web vendor, but couldn't find a satisfactory off-the-shelf solution. We knew that WordPress could do much more for them than other solutions, if we were able to locate a theme and plugins that provided the needed functionality with minimal custom development. Even ifa premium theme was chosen, the low cost would provide technical support, and the open source licensing would allow the group to continue using the solution without ongoing payments. With its ease of use and fast learning curve, WordPress would allow volunteers to help maintain content without mastering professional skills.

The Obstacle — Bias for Print Media

The group was not savvy with digital web processes, and they just proceeded as if it were another print project. Their registration form allowed artists to pick multiple categories in an unstructured way that was confusing, and in the printed directory artists were organized by arbitrary site numbers, so categories were just part of the descriptions. When we determined that each artist should be placed in a primary category to allow for filtering by category for the sake of the user interface, there were arguments about that. Eventually, the group agreed that categories could be used to organize the artists online, as long as site numbers were also included. Another problem was getting art image submissions from all the artists for their profile. The graphic designer only used a small amount of artworks for the limited space on the printed materials, and submission of art images was not included in the registration process. We needed to add a follow-up process to the registration system to request that artists provide animage, and we had to optimize an image for every artist, rather than pick a small number of the best submissions to work with.

The Solution — A Robust Plugin/Theme Stack

We surveyed the field of WordPress plugins and themes to find a solution for a robust directory system with a variety of listing types with geo mapping automation for site addresses. We found free plugins that could be combined to provide custom post types with geo mapping, but even with developer documentation, they required a lot of development to createa complete system. We found an excellent premium theme, Listable by Pixel Grade based upon the popular WP Job Manager stack of Word Press plugins. WP Job Manager is a free job listing plugin by Automattic for basic job-board functionality that is complemented by an array of free and premium addons that allow it to be extended in many ways. The Listable theme combines a slick UX with customizable listings and geo mapping. It was easily affordable, and best of all, the developers at Pixel Grade provide excellent documentation and support.WP Job Manager and related plugins were used to automate the loading of listings by using spreadsheets of artist registrations and sponsor information to import listings in batches. Rather than spend time cutting and pasting and retyping all the directory content, a volunteer content manager focused their time on minimal updates and mostly adding value with news posts and social media work.

The Result — LolaArtCrawl.com

While the solution needed some customization for the particular needs of this project, the comprehensive theme/plugin stack provided about 90% of what was needed, including responsive mobile views and interactive mapping for all the artist sites and business sponsors. We met the goals for the project to improve the web outreach with new features that enhanced the event:

- Every artist had a profile with description and an image of their artwork
- Every business sponsor had a profile that rotated through the artist listings for promotion
- Art crawl attendees were able to find and navigate to sites on smart phones
- A volunteer content manager posted news updates on the website with minimal training
- News updates were optimized for social media sharing
- Social media tweets were automatically embedded into the website
- The website can be reused next year by uploading new graphics and listings into the theme

4.7 CASE STUDY ON GCC

Programming Languages Supported by GCC:

GCC stands for "GNU Compiler Collection". GCC is an integrated distribution of compilers for several major programming languages. These languages currently include C, C++, Objective-C, Objective-C++, Fortran, Ada, D, Go, and BRIG (HSAIL).The abbreviation GCC has multiple meanings in common use. The current official meaning is "GNU Compiler Collection", which refers generically to the complete suite of tools. The name historically stood for "GNU C Compiler", and this usage is still common when the emphasis is on compiling C programs. Finally, the name is also used when speaking of the language-independent component of GCC: code shared among the compilers for all supported languages. The language-independent component of GCC includes the majority of the optimizers, as well as the "back ends" that generate machine code for various processors.

The part of a compiler that is specific to a particular language is called the "front end".

In addition to the front ends that are integrated components of GCC, there are several other front ends that are maintained separately. These support languages such as Mercury, and COBOL. To use these, they must be built together with GCC proper. Most of the compilers for languages other than C have their own names. The C++ compileris G++, the Ada compiler is GNAT, and so on. When we talk about compiling one of those languages, we might refer to that compiler by its own name, or as GCC. Either is correct. Historically, compilers for many languages, including C++ and Fortran, have been implemented as "preprocessors" which emit another high level language such as C. None ofthe compilers included in GCC are implemented this way; they all generate machine code directly. This sort of preprocessor should not be confused with the C preprocessor, which is an integral feature of the C, C++, Objective-C and Objective-C++ languages.

C Language:

The original ANSI C standard (X3.159-1989) was ratified in 1989 and published in 1990.

This standard was ratified as an ISO standard (ISO/IEC 9899:1990) later in 1990. There were no technical differences between these publications, although the sections of the ANSI standard were renumbered and became clauses in the ISO standard. The ANSI standard, but not the ISO standard, also came with a Rationale document. This standard, in both its forms, is commonly known as C89, or occasionally as C90, from the dates of ratification. To select this standard in GCC, use one of the options '-ansi', '-std=c90' or '-std=iso9899:1990'; to obtain all the diagnostics required by the standard, you should also specify '-pedantic' (or '-pedantic-errors' if you want them to be errors rather than warnings). Errors in the 1990 ISO C standard were corrected in two Technical Corrigenda published in 1994 and 1996. GCC does not support the uncorrected version. An amendment to the 1990 standard was published in 1995. This amendment added digraphs and __STDC_VERSION__ to the language, but otherwise concerned the library. This amendment is commonly known as AMD1; the amended standard is sometimes known as C94 or C95. To select this standard in GCC, use the option '-std=iso9899:199409' (with, as for other standard versions, '-pedantic' to receive all required diagnostics). A new edition of the ISO C standard was published in 1999 as ISO/IEC 9899:1999, and is commonly known as C99. (While in development, drafts of this standard version were referred to as C9X.) GCC has substantially complete support for this standard version; see http://gcc.gnu.org/c99status.html for details. To select this standard, use '-std=c99' or '-std=iso9899:1999'. Errors in the 1999 ISO C standard were corrected in three Technical Corrigenda published in 2001, 2004 and 2007. GCC does not support the uncorrected version. A fourth version of the C standard, known as C11, was published in 2011 as ISO/IEC 9899:2011. (While in development, drafts of this standard version were referred to as C1X.) GCC has substantially complete support for this standard, enabled with '-std=c11' or '-std=iso9899:2011'. A version with corrections integrated was prepared in 2017 and published in 2018 as ISO/IEC 9899:2018; it is known as C17 and is supported with '-std=c17' or '-std=iso9899:2017'; the corrections are also applied with '-std=c11', and the only difference between the options is the value of __STDC_VERSION__. A further version of the C standard, known as C2X, is under development; experimental and incomplete support for this is enabled with '-std=c2x'. By default, GCC provides some extensions to the C language that, on rare occasions conflict with the C standard. See Chapter 6 [Extensions to the C Language Family], page 503. Some features that are part of the C99 standard are accepted as extensions in C90 mode, and some features that are part of the C11 standard are accepted as extensions in C90 and C99 modes. Use of the '-std' options listed above disables these extensions where they conflict with the C standard version selected. You may also select an extended version of the C language explicitly with '-std=gnu90' (for C90 with GNU extensions), '-std=gnu99' (for C99 with GNU extensions) or '-std=gnu11' (for C11 with GNU extensions). The default, if no C language dialect options are given, is '-std=gnu17'.

The ISO C standard defines (in clause 4) two classes of conforming implementation. A conforming hosted implementation supports the whole standard including all the library facilities; a conforming freestanding implementation is only required to provide certain library facilities: those in <float.h>, <limits.h>, <stdarg.h>, and <stddef.h>; since AMD1, also those in <iso646.h>; since C99, also those in <stdbool.h> and <stdint.h>; and since C11, also those in <stdalign.h> and <stdnoreturn.h>. In addition, complex types, added in C99, are not required for freestanding implementations. The standard also defines two environments for programs, a freestanding environment, required of all implementations and which may not have library facilities beyond those required of freestanding implementations, where the handling of program startup and termination are implementation-defined; and a hosted environment, which is not required, in which all the library facilities are provided and startup is through a function int main (void) or int main (int, char *[]). An OS kernel is an example of a program running in a freestanding environment; a program using the facilities of an operating system is an example of a program running in a hosted environment.

GCC aims towards being usable as a conforming freestanding implementation, or as the compiler for a conforming hosted implementation. By default, it acts as the compiler for a hosted implementation, defining __STDC_HOSTED__ as 1 and presuming that when the names of ISO C functions are used, they have the semantics defined in the standard. To make it act as a conforming freestanding implementation for a freestanding environment, use the option '-freestanding'; it then defines __STDC_HOSTED__ to 0 and does not make assumptions about the meanings of function names from the standard library, with exceptions noted below. To build an OS kernel, you may well still need to make your own arrangements for linking and startup. GCC does not provide the library facilities required only of hosted implementations, not yet all the facilities required by C99 of freestanding implementations on all platforms. Touse the facilities of a hosted environment, you need to find them elsewhere (for example, in the GNU C library). Most of the compiler support routines used by GCC are present in 'libgcc', but there are a few exceptions. GCC requires the freestanding environment provide memcpy, memmove, memset and memcmp. Finally, if __builtin_trap is used, and the target does not implement the trap pattern, then GCC emits a call to abort. For references to Technical Corrigenda, Rationale documents and information concerning the history of C that is available online, see http://gcc.gnu.org/readings.html

C++ Language:

GCC supports the original ISO C++ standard published in 1998, and the 2011, 2014, 2017 and mostly 2020 revisions. The original ISO C++ standard was published as the ISO standard (ISO/IEC 14882:1998) and amended by a Technical Corrigenda published in 2003 (ISO/IEC 14882:2003). These standards are referred to as C++98 and C++03, respectively. GCC implements the majority of C++98 (export is a notable exception) and most of the changes in C++03. To select this standard in GCC, use one of the options '-ansi', '-std=c++98', or '-std=c++03'; to obtain all the diagnostics required by the standard, you should also specify '-pedantic' (or'-pedantic-errors' if you want them to be errors rather than warnings).A revised ISO C++ standard was published in 2011 as ISO/IEC 14882:2011, and is referred to as C++11; before its publication it was commonly referred to as C++0x. C++11 contains several changes to the C++ language, all of which have been implemented in GCC. For details see https://gcc.gnu.org/projects/cxx-status.html#cxx11. To select this standard in GCC, use the option '-std=c++11'.Another revised ISO C++ standard was published in 2014 as ISO/IEC 14882:2014, and is referred to as C++14; before its publication it was sometimes referred to as C++1y. C++14contains several further changes to the C++ language, all of which have been implemented in GCC. For details see https://gcc.gnu.org/projects/cxx-status.html#cxx14. Toselect this standard in GCC, use the option '-std=c++14'.

The C++ language was further revised in 2017 and ISO/IEC 14882:2017 was published.

This is referred to as C++17, and before publication was often referred to as C++1z. GCC supports all the changes in that specification. For further details see https://gcc.gnu. org/projects/cxx-status.html#cxx17. Use the option '-std=c++17' to select this variant of C++.

Another revised ISO C++ standard was published in 2020 as ISO/IEC 14882:2020, and is referred to as C++20; before its publication it was sometimes referred to as C++2a. GCC supports most of the changes in the new specification. For further details see https:// gcc.gnu.org/projects/cxx-status.html#cxx20. To select this standard in GCC, use the option '-std=c++20'.More information about the C++ standards is available on the ISO C++ committee's website at http://www.open-std.org/jtc1/sc22/wg21/.To obtain all the diagnostics required by any of the standard versions described above you should specify '-pedantic' or '-pedantic-errors', otherwise GCC will allow somenon-ISO C++ features as extensions. By default, GCC also provides some additional extensions to the C++ language that on rare occasions conflict with the C++ standard. See Section 3.5 [C++ Dialect Options], page 50. Use of the '-std' options listed above disables these extensions where they they conflict with the C++ standard version selected. You may also select an extended version of the C++ language explicitly with '-std=gnu++98' (for C++98 with GNU extensions), or '-std=gnu++11' (for C++11 with GNU extensions), or '-std=gnu++14' (for C++14 with GNU extensions), or '-std=gnu++17' (for C++17 with GNU extensions), or '-std=gnu++20' (for C++20 with GNU extensions).

The default, if no C++ language dialect options are given, is '-std=gnu++17'

4.8 CASE STUDY ON GDB

What is gdb?

It is a "GNU Debugger". A debugger for several languages, including C and C++. It allows you to inspect what the program is doing at a certain point during execution. Errors like segmentation faults may be easier to find with the help of gdb. Most computer systems have one or more debugging tools available. These can save you a tremendous amount of time and frustration in the debugging process. The tool available on almost all Unix systems is gdb.

How to Use gdb:

Easy to Learn: In my own debugging, I tend to use just a few gdb commands, only four or five in all. So, you can learn gdb quite quickly. Later, if you wish, you can learn some advanced commands.

The Basic Strategy: A typical usage of gdb runs as follows: After starting up gdb, we set breakpoints, which are places in the code where we wish execution to pause. Each time gdb encounters a breakpoint, it suspends execution of the program at that point, giving us a chance to check the values of various variables.

In some cases, when we reach a breakpoint, we will single step for a while from that point onward, which means that gdb will pause after every line of source code. This may be important, either to further pinpoint the location at which a certain variable changes value, or in some cases to observe the flow of execution, seeing for example which parts of if-then-else constructs are executed.

As mentioned earlier, another component of the overall strategy concerns segmentation faults. If we receive a ``seg fault'' error message when we exeucte our program (running by itself, not under gdb), we can then run the program under gdb (probably not setting any breakpoints at all). When the seg fault occurs, gdb will tell us exactly where it happened, again pinpointing the location of the error. (In some cases the seg fault itself will occur within a system call, rather than in a function we wrote, in which case gdb'sbt command can be used to determine where in our code we made the system call.) At that point you should check the values of all array indices and pointers which are referenced in the line at which the error occurred. Typically you will find that either you have an array index with a value far out of range, or a pointer which is 0 (and thus unrefenceable). Another common error is forgetting an ampersand in a function call, say scanf(). Still another one is that you have made a system call which failed, but you did not check its return value for an error code.

Invoking/Quitting gdb

Before you start, make sure that when you compiled the program you are debugging, you used the -g option, i.e.

```
cc -g sourcefile.c
```

Without the -g option, gdb would essentially be useless, since it will have no information on variable and function names, line numbers, and so on.

Then to start gdb type

```
gdb filename
```

where `filename' is the executable file, e.g.a.out for your program.
To quit gdb, type `q'.

The r (Run) Command

This begins execution of your program. Be sure to include any command-line arguments; e.g. if in an ordinary (i.e. nondebugging) run of your program you would type

```
a.out< z
```

then within gdb you would type

```
r < z
```

If you apply r more than once in the same debugging session, you do not have to type the command-line arguments after the first time; the old ones will be repeated by default.

The b (Breakpoint) and c (Continue) Commands

This says that you wish execution of the program to pause at the specified line.
For example,

```
b 30
```

means that you wish to stop every time the program gets to Line 30.

Again, if you have more than one source file, precede the line number by the file name and a colon as shown above.

Once you have paused at the indicated line and wish to continue executing the program, you can type c (for the continue command).

You can also use a function name to specify a breakpoint, meaning the first executable line in the function.

For example,

```
b main
```

says to stop at the first line of the main program, which is often useful as the first step in debugging.

You can cancel a breakpoint by using the disable command.

You can also make a breakpoint conditional. E.g.

```
b 3 Z > 92
```

would tell gdb to stop at breakpoint 3 (which was set previously) only when Z exceeds 92.

4.9 CASE STUDY ON GITHUB

GitHub, the easiest way for developers to write software together, has scaled into a collaboration of 5.8 million developers across more than 12 million repositories worldwide. GitHub worked with Fastly to customize their CDN set up, ensuring rapid and efficient delivery of their content. Fastly serves all static assets and sits in front of GitHub.com, Pages (their website hosting service), and raw.github.com.

Open Source Software (OSS) has been adopted not only for the personal purpose software products but also for core systems of companies and public institutions. It becomes indispensable for our society. In order to evolve and grow OSSs, it is important to acquire not only stakeholders but also contributors widely from outside. Especially, in the case of large OSS, a large number of contributors are needed. Therefore, it is extremely important to explore the acquisition mechanism of contributors.

In this research, we try to elucidate the mechanism by analyzing the OSS projects on GitHub. Although GitHub is an Internet hosting service aimed at supporting software collaborative development, there are many other uses that actually do not require contributors, such as free file storage or learning of version control system. To analyze collaborative software development projects, it is necessary to exclude projects other than the purpose.

To exclude unrelated use projects, we pay attention on so called contribution file. GitHub has a common file called contribution file (contributing.md/contributing.txt [1]) to describe the necessary contribution for the project. The file describes the contribution method that the project expects so that the applicant can judge whether the applicant participates by seeing the contents. By extracting the project including this file, we are able to analyze the projects which intend collaborative software development. In this research, 459 projects including contribution file were extracted from a GitHub archive. By analyzing these projects, we try to clarify whether contributors actually decide to participate by checking the file contents. Also we try to analyze what kind of description is important to acquire contributors.

Data Preparation

In this research, we use data of BOA [5] provided by Iowa State University which is a GitHub archive (September 2015 Full) to efficiently discover projects that contain contribution files. The proportion of projects that contain contribution files is extremely low, and if we search directly from the web, it takes a lot of time. BOA is a huge archive of GitHub Project which adopted Hadoop's technology, and it can acquire the target Project and its attributes in a very short time. We searched 7,830,023 projects and found that as of September 2015 only 639 Projects contain contribution file. Since there is no contribution file on the BOA, we obtained them by using the acquired URL, but as of April 2017 there were only 459 files. After all, in 7,830,023 projects, we were able to get only 459 contribution file. (0.006%)

Project attributes are obtained from projects where contribution file exist by using GitHub API. Attributes acquired are mainly items that can be checked on the GitHub web screen such as Stargazer number (Bookmark), Subscriber number and so on. In order to track changes over time, we plan to continue to acquire attributes every month.

4.10 CASE STUDY ON OPEN OFFICE

Introduction:

Open Office.org is an open source multi-platform office suite software. It is built in some software, such as word processing, spreadsheets, presentation tools and graphics software, etc. And it uses C++ as a procedural framework, provides international support and documentation of authorized applications Program interface, has simple, practical, economic and cross-platform features.

In the promotion of Uighur linux operating system platform in Xinjiang region, it needs the Uighur editing and typesetting supporting software because the editorial direction of Uighur is from right to left, but Chinese and English editor direction is in the opposite, and meantime it is a complex textwith the complex characteristics such as character deformation and even pens. Handling and compatible with Uighur in Chinese and English systems need to address Localization technical issues. Therefore, we have carried out the research and development work based on the OpenOffice.org office suite software.

Software Requirements

Modern Uighur letters is formed on the basis of Arabic alphabets, there are 32 [1]. Uighur is a handwritten text, in needs for joint document when writing, letters is in the different location (in the prefix, middle, end of the words), there will be in different writing forms. In the Unicode encoding standard, 32 separate letters are called Character name, and their code bits are assigned in U+0600−U+06FF area. The letters in the prefix, middle, and the end of loactions are called appearing form. Their code bits are assigned in U+FE70-U+ FEFF Arabic extended character region.

There are some biggest difficulties in dealing with Uighur. For example, its opposite editorial direction is different from Chinese and English. And It is difficult in cursor control. Another problem is that the Uighur are deformation text. Its adjacent letters are likely to be deformed when insert or delete a letter. Therefore it is more difficult when handling.

Uighur version of Open Office is the lightweight office software, mainly used for typesetting mixed-edit about Uighur, Chinese and English. It should include all the standard features of Office, such as word processing, slide presentations, spreadsheets, web editing and conversion, multimedia browser and picture viewer, etc. All interfaces should support for right to left editing, and their display styles, interface texts, and help information should be in Uighur. It should be able to fully support the Uighur typesetting rules, including right-aligned, right to left writing rules, letter-shaped auto-selection rules, the rules fit the word deformation show. And support process and display complex document when Integrating Chinese, English and other multilingual word, and when the two styles are mixed up, each text can be edited in their own customary rules. It can offer a variety of standard Uighur OpenType fonts which support the zoom control of Uighur. Realizes the layout, edit control model of Office suite embedded in a web browser, support for multi-lingual e-government and B/S application development. Provide an easy using, graphical system installation and management tools for minority users. Support platform installation in the domestic multi-language Linux system, Chinese Linux system, and other windows series system. Compatible with office suite software, and you can open, edit, and save Microsoft Word, Excel, PowerPoint documents.

Uighur localization achievement:

Uighur localization work starts from the source code, and make the software support Uighur from the ground core of the code, including support from the Uighur code, make Uyghur language production libraries, creating local language-specific resources and information, preparation of Uighur input method, Uighur localization about Interface, editing and typesetting software for Uighur adapter and so on. Mainly by the following steps: (1) add Uighur language to the resource system, (2) add Uighur code to the build environment, (3) add to the localization tools about Uighur; (4) to extract the string from the source file, them into Uighur language and merge the translated string into the source file; (5) add Uighur into the installation module; (6) Add Uighur identity to Localedata.cxx; (7) add the text box option of Uighur localization environment to the the OpenOffice.org Option dialog.

4.11 CASE STUDY ON LICENSING

An Open-Source-Software-based License:

The amount of open-source-software (OSS) used in the global software engineering community is already enormous and still growing. This includes both the products we develop and the development tools we use to create them. It is meanwhile rare to find examples of products that do not contain open source components. Although, using open source components in products does have many advantages, it is very important that one also manages the use of the open source components in a license-compliant way. A set of companies and other organizations who either offer or use OSS-based license compliance tools have recently formed the “Open Source Tooling Group”. This international group works on establishing an ecosystem of OSS-based tools for license compliance that fit together well and can offer an ecosystem of tools for organizations to help fulfill their license compliance obligations. This talk provides the motivation and overview of this topic describing the relevance to software engineering practitioners. It will close by highlighting some of the research areas where further improvements could be done in this fast-growing field.

GNU GPL

The positive aspects of GNU GPL license are

- Maintenance of GPL for derivative software thus guaranteeing free access and distribution.
- Most commonly used free software licensing method.
- The integrator can view and use the source code for his development activities.
- The integrator can modify the source code for his products.
- The integrator can provide his software under his own name.
- The integrator will have a good community support.
- Competitors cannot use the software to create closed-source derived products.

As negative aspects it was mentioned

- SDs has to make their modifications available for all who use their products.
- Libraries wanting to use the SD's product must also be under the GPL, thus possibly limiting acceptance.

Generally we can say, that commercial companies may not want to further develop software under GPL because they can not later change the licensing type and that the integration into closed source or even open-source environments with other licenses can be hard and particularly not be done.

GNU LGPL license

The positive aspects of GNU LGPL license are:

- The integrator can use the software for his products without limitations.
- The integrator can modify the source code for his products without limitations.
- The integrator SD can provide his software under his own name.
- The integrator will have a good community support for the library.
- The integrator can use the product licensed under the LGPL even in proprietary systems.
- The library itself will be updated by the community, since changes have to be contributed back.

As negative aspects it was mentioned

- Permits use of the library in proprietary programs thus enabling the possibility to not access all software that uses the library.

- Competitors might use the software and create concurrence commercial products Generally integrators prefer LGLP license against GPL.

4.12 CASE STUDY ON MODE OF FUNDING

A look at various funding models for open source projects. Most of the major open source projects require a fair amount of development and maintenance and have many full-time people working on them.

People volunteer their time, most of the members of the numerous open source software foundations are unpaid and dedicate their own time and energy. Many of these volunteers work for companies who understand the importance of open source and give them the time.

Here’s a look at how open source projects, which by nature do not charge for their software, generate money to fund their projects.

Donations:

Wikipedia is the best example of the donation model. Annually it turns to its users directly asking for donations, using banner ads and on-site promotions. The Wikimedia Foundation also receives funding from benefactors and grants. Wikimedia follows a very similar model as public radio in the U.S.

Bitcoin Foundation is funded by a membership model which you can join for a donation or if you wish you can donate without membership. Membership is another very common model for offline non-profit organizations, for example the ACLU. The benefits of a membership model allows for recurring revenue and a mailing list you can reach out to to solicit future donations.

Software foundations such as the Free Software Foundation, Software for the Public Interest, Software Freedom Conservancy and Apache Foundation operate under a similar sponsorship model, with the majority of donations coming from large corporate sponsors.

Corporate Sponsor or Patronage:

Companies benefit greatly from open source software and will hire and employ people just to work on them. Google employed creator of Python, Guido van Rossum for 7-years. Yahoo employed creator of PHP, RasmusLerdorf for many years to further PHP development. Two examples which key people are able to dedicate their time.

Linux has long been funded and advanced by corporate contributors. You can see the amount of time given to open source by the Linux contributors list. A majority of these contributions are for drivers to make the company’s hardware work with Linux. The companies are motivated to get involved but the time and code contributed is still open source.

Android was created by Google and released as open source to generate a platform audience, receive contributions and feedback and encourage adoption. Also, as open source third-party manufacturers are more willing to adopt a platform since it is open. Google still maintains primarily control and employs the majority of developers.

Ruby on Rails is another example with 37 Signals releasing the software to both give back to the community, but also with larger adoption, they benefit through feature development and bug fixes. Also, as a company an increase in both community goodwill and experienced engineers; making it easier to hire. You can see the core team of Rails has since expanded beyond 37 Signals.

Other examples of companies developing then releasing software as open source include, Java by Sun Microsystems, Cassandra by Facebook, Hadoop by Yahoo, Bootstrap by Twitter, V8 and Go programming language by Google. Plus countless libraries released by all sizes of companies.

Commercial Enterprise Support

RedHat was one of the first companies who attempted to build a for-profit business off open source. The RedHat business model is to develop an enterprise version of the Linux platform and offer long-term stability and enterprise support contracts. RedHat had some bumps along the way but has worked out a pretty good model, now working well with the community and is a profitable company with a \$10b valuation # Ubuntu follows a similar model as RedHat but started with more a focus on the consumer market. Canonical, the company behind Ubuntu, was jump started by its founder Mark Shuttleworth who self-funded it. Canonical offers support and services to enterprises and governments. However it continues to struggle between profitability and ambitious projects, such as its recent failures with the Ubuntu Edge device and Ubuntu One file services.

MySQL started and continues to be a dual-licensed product, originally run by a Swedish-company MySQL AB, but now run by Oracle. The two versions of MySQL are an open source community version licensed under the GPL and a commercial enterprise server which includes support and advanced features not available in the community edition, for example backup, monitoring and high-availability services. Sun Microsystems acquired MySQL for \$1b in 2008.

Virtual Box is an open source virtualization tool which Oracle releases the basic app free for all. They also offer advanced features and extensions which are free for personal use, but requires a commercial license for business use. This model appears to work well to encourage adoption, for example the Vagrant project uses VirtualBox as its primary engine.

Zend Framework is a PHP Framework used to develop web applications. The primary framework is open sourced and Zend Technologies gives away to encourage adoption. They sell their Zend Server product which adds additional features for packaging, deployment and support.

4.13 CASE STUDY ON COMMERCIAL/NON-COMMERCIAL USE

Non-Commercial use means open source software use.

Open source Software: Open source software is the computer software developed either by an individual, group or an organization to meet certain requirements and it is available for any modifications based on its developing body’s interest. Open source software is published openly for general public and here the source code is open for all. For open source software the users do not need to spend any cost. It is available under free licensing. It depends on donations and support as its main source of find.

Some examples of open source software are Firefox, Open Office, Zimbra, VCL media player, Thunderbird.

- Open source software is the computer software developed either by an individual, group or an organization to meet certain requirements and it is available openly for general public for any modifications based on its developing body's interest.
- The cost of open source software is free.
- Open source provides limited technical support.
- Open source software is available under free licensing.
- In open source software users need to rely on community support.
- In open source software installation and updates are administered by the user.
- Limited hands on training and online resources are available for open source software application training.
- Here in open source software users can customize.
- In this rapid community response helps in fixing the bugs and malfunctions.
- In open source software the source code is public.
- The source of funds of open source software mainly depends on donations and support.

Commercial Software: Commercial software is the computer software where only the person, team, or organization that created it can modify also they have exclusive right over the software. Anyone needs to use it has to pay for it valid and authorized license. Here the source code is protected. For commercial software the users need to spend moderate to expensive cost. It is available under high licensing cost. It depends on its software sale / product licensing as its main source of fund.

Some examples of commercial software are Windows Operating System, MS Office, SAP, Oracle, Adobe Photoshop.

- Commercial software is the computer software where only the person, team, or organization that created it can modify also they have exclusive right over the software. Anyone needs to use it has to pay for it valid and authorized license.
- The cost of commercial software varies from moderate to expensive.
- Commercial software provides guaranteed technical support.
- Commercial software is available under high licensing cost.
- In commercial software users get dedicated support from the vendor.
- In commercial software installation and updates are administered by the software vendor.
- On site and Online trainings are available from the commercial software vendor side for software application training.
- But in commercial software mainly vendor offers customization.
- In this mainly the vendor is responsible for fixing the malfunctions.

- In commercial software the source code is protected.
- The source of fond of commercial software depends on its software sale / product licensing.

4.14 CASE STUDY ON OPEN SOURCE HARDWARE

Defining Open Source Hardware:

Simply put, open source hardware is a term that refers to any type of device whose hardware specifications are fully documented or otherwise available.

That's important for several reasons. First, it maximizes the ability of third-party programmers and partners to work with a given device. In most cases hardware manufacturers provide only a basic level of programmability by releasing software development kits (SDKs) or limited documentation about hardware specifications. Sometimes additional hardware information is available through partner programs. But with open source hardware, all information is freely available to the public.

Another reason why hardware openness matters is that it lets users know exactly what their hardware does. If you've ever read stories about webcams spying on users or microphones listening in without their permission, you appreciate the value of being able to know everything your hardware is capable of doing (and how it can be activated), as opposed to knowing only what the company you buy it from reveals.

Open hardware has the benefit of being more extensible, too. For most people this matters with hardware even less than it does with software. But for the hardcore geeks out there who want to be able to customize to infinity, documentation about how hardware works is crucial. It makes it much easier to tweak a device by cutting wires, plugging in additional components and so on.

Open Source Hardware Origins

If the benefits of open hardware sound a lot like the ones you get from open source software, it's because they are. And the relationship between open hardware and open source software is not incidental.

As a conscious movement, open hardware dates to the late 1990s, when Bruce Perens announced an open hardware certification program that had the backing of a number of industry partners (most of them were companies that sold Linux hardware, software or support services). Less than a year later, Perens was also one of the figures who helped launch the open source software movement properly defined.

Yet in practice, open hardware goes back much further. Like open source code, open hardware specifications were the default during the first decades of computing. At that time, when many programs were written in assembly code and software was much less portable than it is today, intricate knowledge of hardware was essential for writing software. That meant that companies that manufactured hardware were much more forthcoming than they generally are today with hardware documentation.

The shift toward closed-source software starting in the early 1980s, combined with the standardization of basic hardware platforms like the IBM PC and the adoption of cross-platform programming languages such as C, made hardware specifications less important. For the most part, programmers no longer needed to know lots of details about hardware specifications in order to write code for a particular platform. As long as you wrote for the PC, your code would run on most computers. And when hardware-specific software was required, companies could release it in closed-source form, which did not require them to give away details about the hardware.

Why Open Hardware Matters Today

Closed hardware remained the norm as the PC age gave way to the era of mobile devices and the cloud. For the most part, only tinkerers and DIYers had reason to wish hardware were more open. For ordinary users, open hardware has not traditionally offered many advantages.

But open hardware is poised to assume more importance going forward. This is due in part to the influence of open source software, which has now become predominant. As organizations come to expect all software source code to be open in order to maximize interoperability, it's only natural for them to think the same way about hardware.

Open hardware will also matter on the IoT, for two main reasons. The first involves security and privacy. While worries about snooping webcams on PCs may be overblown, demands for privacy assurances will reach new magnitudes when IoT devices surround consumers and collect all sorts of personal information. Companies that build IoT solutions based on open hardware will be able to make privacy promises that others can't.

Open hardware will also help to drive IoT adoption by creating a foundation for building low-cost, portable IoT solutions. In other words, open hardware platforms, like Arduino, will do for IoT what open source software platforms, like Linux and Apache, did for the Web by providing a convenient, accessible, cost-efficient basis on which to deploy products in a new ecosystem.

Last but not least, the software-defined revolution is likely to increase demand for open source hardware. That's because software-defined solutions abstract functions like networking and storage infrastructure from the underlying devices, making hardware itself less valuable. Organizations will have no need for expensive, proprietary network switches or storage arrays when such hardware cannot do anything that can't be done in software alone.

Against this backdrop, open hardware solutions will become more valuable not only because they are likely to be inexpensive, but also because open specifications maximize the ability of programmers to take advantage of hardware features when optimizing software-defined solutions.

4.15 CASE STUDY ON OPEN SOURCE DESIGN

What is an open source design?

Open Source Design is the development of technology and ideas without retention of intellectual property. The goal of the movement is to allow for the continued development and full customization of products. Typically, this is done for software; however, it is increasingly being done for hardware. Notable examples include the Tesla car and Arduino. Some advantages of Open Source are its low cost and customization; furthermore, open source platforms tend to expand markets. The major drawback of this is that it is not fully unified. Programs may be incomplete, resulting in bugs, and hardware may not be well documented. This is being remedied through the internet as communities specific to a project are filling in the blanks.

How it works

As a designer and an advocate for open source, I decided to submit a proposal for one of the gratis projects, to design an AntennaPod 2 logo and icon. The project was posted on May 26, 2020, with a submission deadline of June 27, 2020. AntennaPod is an open source Android podcast manager, and the project is looking for a logo that doesn't look similar to the many other podcast logos (e.g., radiating antennas, microphones, sine waves, and such). The organizers provide a very detailed design brief that clearly states their requirements. Even though the project is unpaid, they quickly received replies from several designers.

After the submission deadline, AntennaPod will select three designs and then put them up for a vote by the community.

Design opportunities in open source communities

For designers looking to share their talents and skills with the open source community, Open Source Design is a good place to find opportunities. The same is true for developers and organizations looking for talent to create or improve their projects' design, branding, and user interface.

Designers have many other opportunities to get involved in open source. For example, several months ago, The Document Foundation was looking for logo submissions for its 20th anniversary. This was also a gratis project that I submitted an idea for.

These types of opportunities abound, so keep scouring open source communities for ways you can contribute. You might not get rich, but you may get the glory, and that makes it all worth it!

4.16 CASE STUDY ON OPEN SOURCE TEACHING

What is Open Source Learning?

Open Source Learning is an emerging philosophy of education for the Digital Age. In their day, educational theories such as Waldorf, Montessori, and Reggio Emilia met the evolving needs of learners. Today's challenges and opportunities demand more.

Open Source Learning empowers students to work in partnership with teachers to develop their own learning experiences and interdisciplinary paths of inquiry. Open Source Learning enables students to amplify and accelerate their learning by participating in virtual networks and online communities. Students can apprentice with expert mentors and collaborate with partners around the world. They can also create their own knowledge capital as they learn. Open Source Learning empowers us to produce value, interdependence, and hope in real time.

Why Open Source Learning?

In spite of all the attention and money devoted to improving our education system, today’s learners remain poorly understood and badly underserved. Students, parents, and teachers are suffering. And there is no “one size fits all” solution.

The good news is that we have the answer. Rather than advocating a specific system or required set of techniques, tools, and ideas, Open Source Learning embraces the idea that everyone learns different things in different ways, and it values diverse approaches to reaching those goals. We can now connect beyond the classroom with the ideas, resources, and people that can help students on their learning journeys.

Open Source Learning: methods and tools

In Open Source Learning, each student works with the guidance of a teacher-mentor to develop an interdisciplinary learning journey around a big idea or question.

Students design their experience by working with a variety of tools – especially resources in the digital realm, such as credible internet content, online organizing tools (calendar, project management, and collaboration), 3-D printing, and mobile devices.

As their projects develop, students deepen their experience through information-sharing that establishes themselves and empowers others.

Results

The result of Open Source Learning is that, in the process of mastering concepts and skills, we develop our mental, physical, civic, spiritual, and technical fitness, and we learn new ways to think. For example:

- Open Source Learning allows students to create and manage interactive learning material that becomes available online to everyone, generating and sharing value that extends beyond the traditional K-16 curriculum.
- Deeper and more engaged involvement results in significant improvement in academic achievement.
- Open Source Learning also creates opportunities for traditional performance evaluation of objective production, including formative and summative tests; and alternative assessment of portfolios, which can include a variety of artifacts, including trans media presentation of content, and the student’s choices related to platforms, media, and design.
- Participants – students and teachers alike — emerge with progressively masterful records of assessment and authentic work portfolios that tell a far more compelling story than diplomas or résumés.

4.17 CASE STUDY ON OPEN SOURCE MEDIA

Open source technology is a means of developing computer software through a more collaborative approach than most traditional software. When software is considered "open source," all or part of its source code is made available to the public or purchasers of the software. This allows programmers to modify or augment the software for their needs or the needs of others.

However, "open source" is a term with varying degrees of "openness." At its most basic, it implies that modifications can be made. Going a step further, it can also mean that users are kept up to date by the creators on the development of the software and can influence the choices made. It also depends on the terms of the licensing agreement as to whether or not modified versions of the software can be sold commercially.

Developing open source software boasts a number of advantages for developers and customers. Since the source code is available for all to see, this allows both developers and users to search for bugs. This potentially allows for a more stable, well-tested product. Open source allows for more ideas to be pooled together, making for quicker refinement and innovation. It also helps develop brand loyalty, as users have a hand in the product development and become invested in its success. Most importantly, open source software is frequently far less expensive to develop and purchase.

The downsides of open source software are few but notable. If a company is especially concerned with how the modification of its software may affect its image should avoid open source. Allowing everyone to see source code means that competitors will see the source code. While this makes for good competition, it can also make for copycats developing inferior products aping the original. As far as users are concerned, caution should be exercised when utilizing some open source software, as there may be few resources for technical support when a problem arises.

For smartphones and tablets, the most notable example of open source technology is Google's Android OS. The open source nature of the OS has allowed wireless service carriers and retailers to customize the models they sell. The Google Play Store is especially notable for its wide variety of user-generated applications, which are largely made possible by the OS's open source flexibility.

For PC users, the Linux operating system has been a long-standing example of open source software. Originally released in 1991, it has always been a favorite of the technologically-inclined for its malleability and potential for deep, specialized customization. Modified versions of Linux can even be distributed commercially. The previously mentioned Android OS is actually based off of the Linux framework, which makes the open source nature very fitting.

Summary

In this chapter we learn about Open-source Operating system, various types of open-source software, Open-source concepts like Licensing and funding for open-source software and open-source physical components.

Reference:

https://www.linux.org/
https://www.android.com/
https://github.com/

❖❖❖❖

Unit -5

UNDERSTANDING OPEN SOURCE ECOSYSTEM

Unit Structure

5.0 FREE BSD

What is FreeBSD?

FreeBSD is an operating system for a variety of platforms which focuses on features, speed, and stability. It is derived from BSD, the version of UNIX® developed at the University of California, Berkeley. It is developed and maintained by a large community.

Cutting edge features

FreeBSD offers advanced networking, performance, security and compatibility features today which are still missing in other operating systems, even some of the best commercial ones.

Powerful Internet solutions

FreeBSD makes an ideal Internet or Intranet server. It provides robust network services under the heaviest loads and uses memory efficiently to maintain good response times for thousands of simultaneous user processes.

Advanced Embedded Platform

FreeBSD brings advanced network operating system features to appliance and embedded platforms, from higher-end Intel-based appliances to ARM, PowerPC, and MIPS hardware platforms. From mail and web appliances to routers, time servers, and wireless access points, vendors around the world rely on FreeBSD's integrated build and cross-build environments and advanced features as the foundation for their embedded products. And the Berkeley open source license lets them decide how many of their local changes they want to contribute back.

Run a huge number of applications

With over 33,000 ported libraries and applications, FreeBSD supports applications for desktop, server, appliance, and embedded environments.

Easy to install

FreeBSD can be installed from a variety of media including CD-ROM, DVD, or directly over the network using FTP or NFS.

FreeBSD is free

While you might expect an operating system with these features to sell for a high price, FreeBSD is available free of charge and comes with the source code.

Contributing to FreeBSD

It is easy to contribute to FreeBSD. All you need to do is find a part of FreeBSD which you think could be improved and make those changes (carefully and cleanly) and submit that back to the Project by means of a bug report or a committer, if you know one. This could be anything from documentation to artwork to source code.

2.1 OPEN SOLARIS

Introduction

Open Solaris is an open source operating system, similar in scope to GNU/Linux and BSD, but descended from the proprietary Solaris operating system from Sun Microsystems. The authors of this book find it helpful to think of Open Solaris as divided into three distinct but related aspects: the code, the distributions, and the community.

Features of Solaris

Security: Solaris includes some of the world's most advanced security features, such as Process and User Rights

Management: Trusted Extensions for Mandatory Access Control, the Cryptographic Framework and Secure by Default Networking that allow you to safely deliver new Solutions consolidate with security and protect mission-critical data

Performance: Solaris delivers indisputable performance advantages for database, Web, and Java technology-based services, as well as massive scalability, shattering world records by delivering unbeatable price/ performance advantages.
Networking: With its optimized network stack and support for today’s advanced network computing protocols, Solaris delivers high-performance networking to most applications without modification.

Data Management: Solaris offers dramatic advances in file system and volume management by delivering virtually unlimited capacity and near-zero administration

Interoperability: Understanding that businesses today rely on a mix of technologies from a variety of vendors, Solaris provides tools to enable seamless interoperability with hundreds of heterogeneous hardware and software platforms

Observability: The Solaris release gives you Observability into your system with tools such as Solaris Dynamic Tracing (DTrace), which enables real-time application debugging and optimization

Platform Choice: Solaris is fully supported on more than 900 SPARC-based and x64/x86-based systems from top manufacturers, including systems from Sun, Dell, HP, and IBM

Virtualization: The Solaris OS includes industry-first virtualization features such as Solaris Containers, which let you consolidate, isolate, and protect thousands of applications on a single server

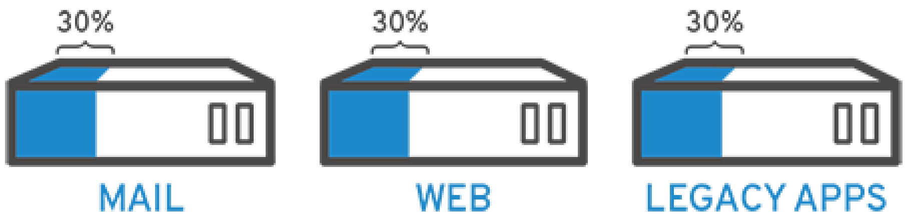
Availability: Solaris features, such as Predictive Self-Healing, support automatic diagnosis and recovery from hardware and application faults, maximizing system uptime.

Support & Services: Offering a broad portfolio of world-class services, Sun can help you extract maximum value from the Solaris Operating System.

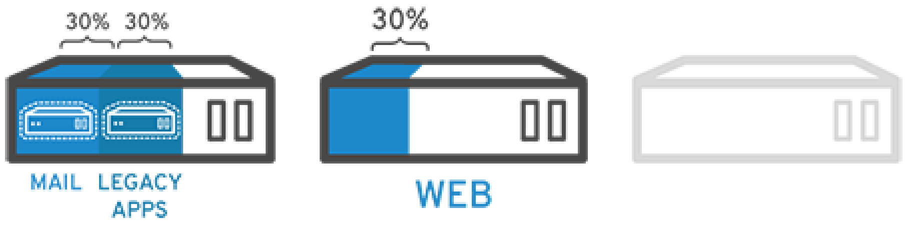
5.2 VIRTUALIZATION TECHNOLOGIES

Virtualization is technology that lets you create useful IT services using resources that are traditionally bound to hardware. It allows you to use a physical machine's full capacity by distributing its capabilities among many users or environments.

In more practical terms, imagine you have 3 physical servers with individual dedicated purposes. One is a mail server, another is a web server, and the last one runs internal legacy applications. Each server is being used at about 30% capacity—just a fraction of their running potential. But since the legacy apps remain important to your internal operations, you have to keep them and the third server that hosts them, right?



Traditionally, yes. It was often easier and more reliable to run individual tasks on individual servers: 1 server, 1 operating system, 1 task. It wasn't easy to give 1 server multiple brains. But with virtualization, you can split the mail server into 2 unique ones that can handle independent tasks so the legacy apps can be migrated. It's the same hardware, you're just using more of it more efficiently.



Keeping security in mind, you could split the first server again so it could handle another task—increasing its use from 30%, to 60%, to 90%. Once you do that, the now empty servers could be reused for other tasks or retired altogether to reduce cooling and maintenance costs.

Below are some examples of open-source Virtualization Technologies

KVM

Short for Kernel-based Virtual Machine, KVM is not as widely deployed as other open source hypervisors, but its stature is growing rapidly.

KVM is a full virtualization hypervisor and can run both Windows and Linux guests.

With the kernel component of KVM included in Linux since kernel 2.6.20, KVM can claim a good level of integration with the rest of the operating system.

KVM received its biggest validation in late 2008 when Linux vendor Red Hat acquired KVM developer, Qumranet. Red Hat now bases its enterprise virtualization server on the KVM hypervisor.

URL: http://www.linux-kvm.org

License: GPL

Xen

Xen began life as a Microsoft-funded startup at the University of Cambridge and has risen to become the "de facto standard" in Linux hypervisors.

Xen supports paravirtualization and "hardware assisted" virtualization for modified and un-modified guests, respectively.
Guests can be Linux or Windows, but the overwhelming majority of guests are Linux variants, particularly in the hosting space.

A few years ago quite a few commercial software vendors, including Novell and Oracle, adopted Xen and then -- seemingly out of nowhere -- the commercial startup behind Xen, XenSource, was acquired by Citrix. Citrix has been Xen-happy ever since.

Recently, CIO reported on the private cloud development at the ACMA in Canberra, which is based on Citrix's Xen hypervisor.
URL: http://www.xen.org
License: GPL

OpenVZ

OpenVZ is container-based virtualization for Linux, which has become quite popular among the mass-market Linux hosting providers as an inexpensive way to provide virtual private servers.

The OpenVZ containers provides the same services as a separate host and claims to provide near native performance.

OpenVZ is the core within Parallels Virtuozzo Containers, a commercial virtualization solution offered by Swiss company Parallels. Commercial support is available for Parallels.

Not a lot has been written about OpenVZ/Parallels in the enterprise space, but there are quite a few glowing user testimonials about the product.

URL: http://openvz.org

License: GPL

VirtualBox

VirtualBox is an open source desktop virtualization tool originally developed by German company, innotek, which was acquired by Sun Microsystems in February 2008.

Since acquiring Sun, Oracle has continued VirtualBox development and the latest version, 4.0, was released in December 2010.

VirtualBox runs on Windows, Linux, Solaris and Mac OS X and can support all those operating systems as guests.

While it is mostly used on desktops, VirtualBox is a full virtualization app and can be used on servers as well.

The closed-source edition of VirtualBox is now distributed as an "extension pack" and includes features like RDP and USB support.

URL: http://www.virtualbox.org
Licence: GPL & CDDL

Lguest

Lguest is an interesting virtualization project started by Australian developer, Paul "Rusty" Russell.

Designed with Linux in mind, lguest allows multiple copies of the same kernel to run alongside each other.

While not a full virtualization hypervisor, lguest prides itself on ease of use and uses the same kernel image for host and guest operating systems.

Computerworld has run with a number of articles about lguest over the past few years.

There's not much information about whether lguest is being used in a business production environments, but that would be interesting.

URL: http://lguest.ozlabs.org/

Licence: GPL

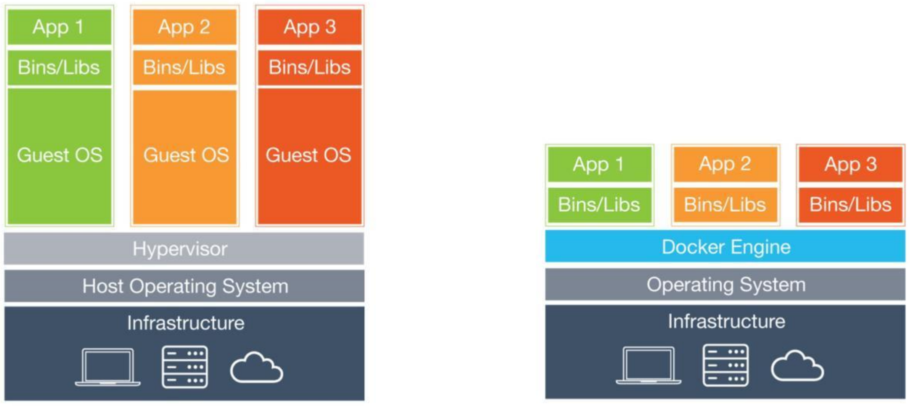
5.3 CONTAINERIZATION TECHNOLOGIES

Docker is a software platform for building applications based on containers — small and lightweight execution environments that make shared use of the operating system kernel but otherwise run in isolation from one another. While containers as a concept have been around for some time, Docker, an open source project launched in 2013, helped popularize the technology, and has helped drive the trend towards containerization and microservices in software development that has come to be known as cloud-native development.

What are containers?

One of the goals of modern software development is to keep applications on the same host or cluster isolated from one another so they don't unduly interfere with each other’s operation or maintenance. This can be difficult, thanks to the packages, libraries, and other software components required for them to run. One solution to this problem has been virtual machines, which keep applications on the same hardware entirely separate, and reduce conflicts among software components and competition for hardware resources to a minimum. But virtual machines are bulky—each requires its own OS, so is typically gigabytes in size—and difficult to maintain and upgrade.

Containers, by contrast, isolate applications’ execution environments from one another, but share the underlying OS kernel. They’re typically measured in megabytes, use far fewer resources than VMs, and start up almost immediately. They can be packed far more densely on the same hardware and spun up and down en masse with far less effort and overhead. Containers provide a highly efficient and highly granular mechanism for combining software components into the kinds of application and service stacks needed in a modern enterprise, and for keeping those software components updated and maintained.



What is Docker?

Docker is an open source project that makes it easy to create containers and container-based apps. Originally built for Linux, Docker now runs on Windows and MacOS as well. To understand how Docker works, let’s take a look at some of the components you would use to create Docker-containerized applications.

Dockerfile

Each Docker container starts with a Dockerfile. A Dockerfile is a text file written in an easy-to-understand syntax that includes the instructions to build a Docker image (more on that in a moment). A Dockerfile specifies the operating system that will underlie the container, along with the languages, environmental variables, file locations, network ports, and other components it needs—and, of course, what the container will actually be doing once we run it.

Docker image

Once you have your Dockerfile written, you invoke the Docker build utility to create an image based on that Dockerfile. Whereas the Dockerfile is the set of instructions that tells build how to make the image, a Docker image is a portable file containing the specifications for which software components the container will run and how. Because a Dockerfile will probably include instructions about grabbing some software packages from online repositories, you should take care to explicitly specify the proper versions, or else your Dockerfile might produce inconsistent images depending on when it’s invoked. But once an image is created, it’s static.

5.4 DEVELOPMENT TOOLS

Docker:

The first and still most popular container technology, Docker's open-source containerization engine works with most of the products that follow, as well as many open-source tools.

Docker Enterprise

This set of extensions not only adds features to Docker, but also makes it possible for Docker (the company) to add commercial support. If you need a support matrix to know exactly which versions of what software are supported—and a phone number to call if things go wrong—then Docker Enterprise might be for you.

CRI-O

The first implementation of the Container Runtime Interface, CRI-O is an incredibly lightweight, open-source reference implementation.

rktlet

The aforementioned rkt, redesigned and retooled to use the CRI as rktlet, now has a set of supported tools and community to rival Docker.

containerd

A project of the Cloud Native Computing Foundation, containerd was an early container format. More recently the developers of containerd built a CRI plugin that lets Kubernetes run containerd in the same way it runs rktlet or CRI-O.

Microsoft Containers

Positioned as an alternative to Linux, Microsoft Containers can support Windows containers under very specific circumstances. They generally run in a true virtual machine and not in a cluster manager like Kubernetes.

5.5 IDES

What is Container Management Software?

Container management platforms facilitate the organization and virtualization of software containers, which may also be referred to as operating-system-level virtualizations. Developers use containers to launch, test, and secure applications in resource-independent environments. Containers house components of applications, libraries, or groups of source code that can be executed on demand. The management platforms help users allocate resources to optimize efficiency and balance system workloads. Containers provide a flexible, portable platform to organize, automate, and distribute applications. Companies use container management software to streamline container delivery to avoid the complexities of interdependent system architectures. The tools are scalable and can greatly improve the performance of widely distributed applications.

Amazon Elastic Container Service (Amazon ECS)

Amazon EC2 Container Service (ECS) is a container management service that supports Docker containers and allows users to easily run applications on a managed cluster of Amazon EC2 instances.

Mirantis Kubernetes Engine (formerly Docker Enterprise)

Mirantis Kubernetes Engine (formerly Docker Enterprise) is the fastest way to modern apps at enterprise scale. Mirantis Kubernetes Engine is the industry-leading and only container platform providing a simple, as-a-service experience and a central point of collaboration across dev and ops to build, share and run modern applications. Schedule a live demo at: www.mirantis.com/demo

Google Kubernetes Engine (GKE)

Deploy, manage, and scale containerized applications, powered by Kubernetes

AWS Fargate

AWS Fargate is a technology for Amazon ECS and EKS that allows you to run containers without having to manage servers or clusters. AWS Fargate removes the need for you to interact with or think about servers or clusters.

Kubernetes

Kubernetes is a Linux container management tool.

IBM Cloud Kubernetes Service

Advanced capabilities for building cloud-native apps, adding DevOps to existing apps, and relieving the pain around security, scale and infrastructure management.

Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) is a solution that optimizes the configuration of popular open-source tools and technologies specifically for Azure, it provides an open solution that offers portability for both users containers and users application configuration.

Portainer

Portainer is the definitive open source UI for simplifying Kubernetes, Docker, Swarm, and ACI container management. Simplicity without compromise - run and manage your complex Kubernetes and Docker environments in a simple, low code/no code manner. Get the best of both worlds - some developers support Kubernetes, while others prefer Docker Swarm or ACI for container orchestration. Portainer helps manage all these environments from a single product. Let devs be devs - developers and DevOps don't always blend. With Portainer's easy low code/no code UI, DevOps can quickly deploy apps without having in-depth knowledge of Kubernetes or Docker Swarm.

Rancher

Rancher is an open-source platform for managing containers that provides a full set of infrastructure services for containers, including networking, storage services, host management and load balancing, work across any infrastructure, and make it simple to reliably deploy and manage applications. Check out our Kubernetes Online Master Class Training Series: https://rancher.com/kubernetes-master-class/

LAMP is an open source Web development platform that uses Linux as the operating system, Apache as the Web server, MySQL as the relational database management system and PHP as the object-oriented scripting language. (Sometimes Perl or Python is used instead of PHP.)

Because the platform has four layers, LAMP is sometimes referred to as a LAMP stack. Stacks can be built on different operating systems. Developers that use these tools with a Windows operating system instead of Linux are said to be using WAMP; with a Macintosh system, MAMP; and with a Solaris system, SAMP.

What is the LAMP Stack?

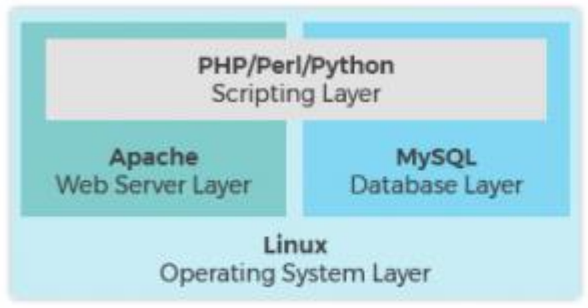
The widely popular LAMP stack is a set of open source software used for web application development.

For a web application to work smoothly, it has to include an operating system, a web server, a database, and a programming language. The name LAMP is an acronym of the following programs:

- Linux Operating System
- Apache HTTP Server
- MySQL database management system
- PHP programming language

Each represents an essential layer of the stack, and together they can create a database-driven and dynamic website.

The illustration below can help visualize how the layers stack together:



Four Components of LAMP Stack
Linux

Linux is the operating system layer and the backbone of the LAMP stack.

All the other components of the stack run on top of this foundation. You can efficiently manage the rest of the stack components on different operating systems such as Windows, macOS, and others. However, Linux has become the front-runner for web development not just because it is open-source, but also due to its flexibility, customization and easy to use technology.

Also, the programming language and database management used in developing a website may dictate the platform you choose to build it on. PHP and MySQL are better suited for Linux. On the other hand, SQL, ASP.NET, and ASP work more efficiently on Windows.

Apache

Apache HTTP Server is a web server software that runs on top of the Linux operating system.

It is the most widely used server, powering more than half of the websites on the internet. The role of the web server is to process requests and transmit information through the internet, using HTTP.

An alternative to Apache is NGINX, a web server whose popularity has been continually increasing since 2008. Whether you go for one or the other depends on what kind of material you want to serve on a webpage, as well as the hosting.

NGINX is a better choice for static content. When it comes to dynamic content, there is a minor difference in performance between the two. Also, Apache is commonly used by shared hosting users, whereas NGINX is mainly used for virtual private servers, dedicated hosting or cluster containers.

MySQL

SQL (Structured Query Language) is the most prevalent query language out there. A query is what we call a request for information or data stored in your database table.

MySQL earned its reputation as an acclaimed database system as it supports SQL and relational tables. By doing so, it makes it much easier to establish dynamic enterprise-level databases.
Consider MySQL if you:

- Need to change the content of your website often
- Have a lot of user-contributed content
- Rely on user feedback
- Have a lot of content that needs to be stored and easily retrieved

Another relational database management system that can be part of the LAMP platform is MariaDB. Both are quite similar, and MariaDB claims to be completely compatible with MySQL, allowing users to transfer their database without any complications or losses. Deciding between the two comes down to whether you feel more comfortable storing data with a large corporation (MySQL under the direction of Oracle Corp) or a completely open source solution (MariaDB).

PHP

PHP (Hypertext Preprocessor) is a programming language which has the role of combining all the elements of the LAMP stack and allowing the website or web application to run efficiently. In short, when a visitor opens the webpage, the server processes the PHP commands and sends the results to the visitor’s browser.

PHP is the fourth layer of the original stack because it interacts exceptionally well with MySQL. It is commonly used for web development because it is a dynamically typed language, making it fast and easy to work with. This feature may be especially appealing if you are a beginner. The reason why PHP is so convenient to use is that it can be embedded into HTML enabling to jump in and out of it as you wish.

In the LAMP stack, the P can also refer to two other programming languages – Perl or Python. All three are simple, yet useful, dynamic tools for creating environments in which you can successfully develop applications. Nowadays, there is a wide variety of scripting languages to choose from, including JavaScript, Ruby, and many more.

Advantages of a LAMP Stack

1. The LAMP stack consists of four (4) components, all of which are examples of Free and Open-Source Software (FOSS). As they are free and available for download, it attracts the attention of many users who wish to avoid paying large sums of money when developing their website.
2. Because it is FOSS, the source code of the software is shared and available for people to make changes and improvements, enhancing its overall performance.
3. The LAMP stack has proven to be a secure and stable platform thanks to its vast community that contributes when any problems arise.
4. What makes it so attractive is that you can easily customize the stack and interchange the components with other open source software to suit your needs.

5.7 OPEN SOURCE DATABASE TECHNOLOGIES

What is open-source database software?

Traditionally, databases have been proprietary tools provided by Oracle, IBM, Microsoft, and a number of other smaller vendors. Over recent years though, and especially for new projects, open source databases and database management tools have steadily grown in maturity and importance. In many cases, open-source database software includes both database software, and the database management tools needed to support the database.

As open-source databases become adopted by more and more companies for large-scale enterprise projects, there has been a concomitant rise in the availability of skilled DBAs, with extensive knowledge of these platforms to be able to assist with mission-critical deployments.

In addition to the obvious cost savings, open source database software have largely reached feature parity with their proprietary cousins. The open-source model also allows for heavy customization and community development, which makes the software very flexible compared to proprietary database software. Training materials are also often provided for free by user communities.

Open-Source Database Software Features & Capabilities

Some of the most common features provided by open-source database software include:

- Relational and Nonrelational Databases
- Support for Multiple Platforms
- Supports databases and database management
- Data Security
- Data Collaboration

Pricing Information

All open-source database software options are available for free to businesses that can support them independently. That said, a number of open-source database options offer paid support, hosting, or monitoring. Pricing depends highly on which features are needed by the organization.

MySQL

MySQL is by far the most popular open-source database out there. Vendors often include it in software packages as the application database.

Even Oracle's own Virtual Machine software runs on MySQL.

MySQL is an open source relational database management system. It is frequently used in webapplications and is even one of the pillars in the LAMP open-source web application software stack (Linux, Apache, MySQL, Perl/PHP/Python). Even websites created in WordPress or Drupal use it as their database.

PostgreSQL

Postgre SQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of Postgre SQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley and has more than 30 years of active development on the core platform.

Postgre SQL has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open source community behind the software to consistently deliver performant and innovative solutions. Postgre SQL runs on all major operating systems, has been ACID-compliant since 2001, and has powerful add-ons such as the popular Post GIS geospatial database extender. It is no surprise that Postgre SQL has become the open source relational database of choice for many people and organisations.

Getting started with using Postgre SQL has never been easier - pick a project you want to build, and let Postgre SQL safely and robustly store your data.

MariaDB Server

MariaDB Server is one of the most popular database servers in the world. It's made by the original developers of MySQL and guaranteed to stay open source. Notable users include Wikipedia, WordPress.com and Google.

MariaDB turns data into structured information in a wide array of applications, ranging from banking to websites. Originally designed as enhanced, drop-in replacement for MySQL, MariaDB is used because it is fast, scalable and robust, with a rich ecosystem of storage engines, plugins and many other tools make it very versatile for a wide variety of use cases.
MariaDB is developed as open source software and as a relational database it provides an SQL interface for accessing data. The latest versions of MariaDB also include GIS and JSON features.

Summary

In this chapter we learn about various open-source operating systems, Open-source hardware, virtualization technologies, open-source databases, IDEs , LAMP.

Reference

https://www.bsd.org/
https://www.docker.com/
http://www.linuxfoundation.org/

❖❖❖❖

University of Mumbai

Dr. Shankar Dayal Sharma Bhavan, Vidyanagri, Santacruz (East), Mumbai 400 098 • www.mu.ac.in

Published by Professor P. A. Mahanwar, Director, IDOL on behalf of University of Mumbai.