KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-641021
(For the candidates admitted from 2015 onwards)
Department of CS,CA & IT

SUBJECT : OPEN SOURCE SOFTWARE
SEMESTER : VI
SUBJECT CODE : 15CSU603A **CLASS : III B.Sc.CS**

COURSE OBJECTIVE:
This course teaches the student the concepts and principles that underlie modern operating systems, and a practice component to relate theoretical principles with operating system implementation. Learn about processes and processor management, concurrency and synchronization, memory management schemes, file system and secondary storage management, security and protection, etc.

COURSE OUTCOME:
- Understand fundamental operating system abstractions such as processes, threads, files, semaphores, IPC abstractions, shared memory regions, etc.
- Understand how the operating system abstractions can be used in the development of application programs, or to build higher level abstractions
- Understand how the operating system abstractions can be implemented
- Understand the principles of concurrency and synchronization, and apply them to write correct concurrent programs/software
- Understand basic resource management techniques (scheduling or time management, space management) and principles and how they can be implemented. These also include issues of performance and fairness objectives, avoiding deadlocks, as well as security and protection.

UNIT I
Overview of Free/ Open Source Software: The Open Source Definition - Examples of OSD Compliant Licenses - Examples of Open Source Software Product – The Open Source Software Development Process – A History of Open Source software: The Berkeley Software Distribution – The Free Software Foundation – Linux – Apache – Mozilla – Open Source Software.

UNIT II
Qualification: Defining Open Source Software – Categorizing Defining Open Source Software – Specific Characteristics of Open Source Software Transformation: The OSS Development Process – Taboos and Norms in OSS Development – The OSS Development Life Cycle –

Deriving a Framework for Analyzing OSS – Zachman"s Framework for IS Architecture – CATWOE and Soft System Method – Deriving the Analytical Framework for OSS.

UNIT III

Environment: The "where" of OSS – the "when" of OSS – World View: A Framework for classifying OSS Motivations – Technological Micro-level (individual) motivation – Economic Micro-level and Macro-level (individual) Motivation – Socio-political Micro-level and Macro-level (individual) Motivation.

Open Source Server Applications: Infrastructure Services – Web Services – Database Servers – Mail Servers – Systems Management – Open Source Desktop Applications: Introduction – Graphical Desktops – Web Browsers – The Office Suite – Mail and Calendar Clients – Personal Software – Cost of OSS: Total Cost of Ownership – Types of Costs- Licensing: Types of Licenses – Licenses in Use – Mixing Open and Close Code – Dual Licensing.

UNIT IV

Perl Programming

Perl - Introduction, Perl Basics: - Syntax, Variables, Strings, Numbers, Operators, Arrays: - Using Arrays, Manipulating Arrays, Associative Arrays, Chop, Length, and Sub string. Hashes, Arguments, Logic, Looping, Files, Pattern Matching, Environment Variables, Using cgilib for Forms.

UNIT V

File Management PERL: - File Handling, Reading From Files, Appending Files, Writing to Files, File Checking, Reading Directories.

Databases PERL: - DBI Module, DBI Connect, DBI Query, MySQL Module, MySQL Connect, MySQL SelectDB, MySQL Query.

TEXT BOOK

1. Kailash vadera, Bhavyesh gandhi, *Open Source Technology*, 2009, Laxmi Publications.
2. Tom Christinasen& Nathan Torkington ,O"Relliy , *Perl CookBook* ,SPD Pvt ltd,2006 Edition.
3. S.Narmadha, V.Raajkumar, *Open source systems*, 2010, Eswar Press.
4. Paul Kavanagh,*Open Source Software: Implementation and Management*, 2004, Digital Press.

.

WEBSITES

1. https://en.wikipedia.org/wiki/Free_Software_Foundation
2. https://en.wikipedia.org/wiki/Open-source_software_development
3. https://timreview.ca/article/146
4. http://www.ijettcs.org/Volume1Issue3/IJETTCS-2012-09-06-008.pdf
5. https://flosshub.org/system/files/p58-feller.pdf
6. https://en.wikipedia.org/wiki/Zachman_Framework
7. https://en.wikipedia.org/wiki/Soft_systems_methodology
8. https://en.wikipedia.org/wiki/Taboo

ESE MARK ALLOCATION

1	**Section A** 20 x 1 = 20	20
2	**Section B** 5 x 8 = 40 Either 'A' or 'B' choice	40
3	TOTAL	60

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-641021

LECTURE PLAN
DEPARTMENT OF COMPUTER SCIENCE

STAFF NAME: D.MANJULA
SUBJECT NAME: OPEN SOURCE SOFTWARE SUB.CODE : 15CSU603A
SEMESTER: VI CLASS : III B.SC CS

S. No	Lecture Duration Period	Topics to be Covered	Support Materials/Page Nos
UNIT I			
1	1	Overview of Free/ Open Source Software	T3: 1-2
2	1	The Open Source Definition	T3: 3
3	1	Examples of OSD Compliant Licenses	T1: 12
4	1	Examples of Open Source Software Product	T1:11
5	1	The Open Source Software Development Process	T1:36-38
6	1	The Open Source Software Development Process[Cont..]	T1:39-42
7	1	A History of Open Source software	T1:16-19
8	1	The Berkeley Software Distribution	T1:75-77
9	1	The Berkeley Software Distribution[Cont..]	T1:78-80
10	1	The Free Software Foundation	W1
11	1	Linux	T1:81-96
12	1	Apache	T1:71-74
13	1	Mozilla	T1: 97-100
14	1	Open Source Software	T3: 3
15	1	Recapitulation and Discussion of Important Questions	
Total Number of Hours planned for Unit I:			15
UNIT II			
1	1	Qualification: Defining Open Source Software	W3
2	1	Categorizing Defining Open Source Software	W3
3	1	Specific Characteristics of Open Source Software Transformation	W4

4	1	The OSS Development Process	T1:36-38
5	1	The OSS Development Process[Cont..]	T1:39-42
6	1	Taboos and Norms in OSS Development	W8
7	1	The OSS Development Life Cycle	W2
8	1	The OSS Development Life Cycle[Cont..]	W2
9	1	Deriving a Framework for Analyzing OSS	W5
10	1	Deriving a Framework for Analyzing OSS[Cont..]	W5
11	1	Zachman‟s Framework for IS Architecture	W6
12	1	Zachman‟s Framework for IS Architecture[Cont..]	W6
13	1	CATWOE and Soft System Method	W7
14	1	Deriving the Analytical Framework for OSS	W5
15	1	Recapitulation and Discussion of Important Questions	
Total Number of Hours planned for Unit II:			15
UNIT III			
1	1	Environment: The “where” of OSS, the “when” of OSS	W5
2	1	World View	W5
3	1	A Framework for classifying OSS Motivations	W5
4	1	Technological Micro-level (individual) motivation	W5
5	1	Economic Micro-level and Macro-level (individual) Motivation	W5
6	1	Socio-political Micro-level and Macro-level (individual) Motivation	W5
7	1	Open Source Server Applications	T4: 145
8	1	Infrastructure Services	T4: 146-147
9	1	Web Services, Database Servers, Mail Servers	T4: 148-167
10	1	Systems Management	T4: 168
11	1	Open Source Desktop Applications: Introduction	T4: 173-174
12	1	Graphical Desktops, Web Browsers, The Office Suite	T4: 175-194
13	1	Mail and Calendar Clients, Personal Software	T4: 195-200
14	1	Cost of OSS: Total Cost of Ownership	T4: 276-284
15	1	Types of Costs Licensing: Types of Licenses, Licenses in Use	T4: 285-287,297-298,298-300,W9
16	1	Mixing Open and Close Code, Dual Licensing.	T4: 300-302
17	1	Recapitulation and Discussion of Important Questions	
Total Number of Hours planned for Unit III:			17

UNIT IV			
1	1	**Perl Programming** Perl - Introduction	W9
2	1	Perl Basics- Syntax, Variables	W9
3	1	Strings, Numbers	T2:20 ,79
4	1	Operators	W9
5	1	Arrays: - Using Arrays	T2:146
6	1	Manipulating Arrays	T2: 148-151
7	1	Associative Arrays	T2: 152-160
8	1	Chop, Length	W10
9	1	Sub string	T2: 72
10	1	Hashes, Arguments	T2: 201
11	1	Logic, Looping	W9
12	1	Files, Pattern Matching	T2: 321, 244
13	1	Environment Variables	W9
14	1	Using cgilib for Forms.	T2: 913
15	1	Recapitulation and Discussion of Important Questions	
Total Number of Hours planned for Unit IV:			15
UNIT V			
1	1	File Management PERL	W9
2	1	File Handling	W9
3	1	ReadingFrom Files, Appending Files	W9
4	1	Writing to Files	W9
5	1	File Checking	W9
6	1	Reading Directories	T2: 215-217, W9
7	1	Databases PERL, DBI Module, DBI Connect, DBI Query	T2:
8	1	MySQL Module, MySQL Connect	W9
9	1	MySQL SelectDB, MySQL Query.	W9
10	1	Recapitulation and Discussion of Important Questions	
11	1	End semester previous question paper	
12	1	End semester previous question paper	
13	1	End semester previous question paper	
		Total Number of Hours planned for Unit V:	13
TOTAL NO.OF HOURS PLANNED : 75			

TEXT BOOK

1. Kailash vadera, Bhavyesh gandhi, *Open Source Technology*, 2009, Laxmi Publications.
2. Tom Christinasen& Nathan Torkington ,O‟Relliy , *Perl CookBook* ,SPD Pvt ltd,2006 Edition.
3. S.Narmadha, V.Raajkumar, *Open source systems*, 2010, Eswar Press.
4. Paul Kavanagh,*Open Source Software: Implementation and Management*, 2004, Digital Press.

WEBSITES

1. https://en.wikipedia.org/wiki/Free_Software_Foundation
2. https://en.wikipedia.org/wiki/Open-source_software_development
3. https://timreview.ca/article/146
4. http://www.ijettcs.org/Volume1Issue3/IJETTCS-2012-09-06-008.pdf
5. https://flosshub.org/system/files/p58-feller.pdf
6. https://en.wikipedia.org/wiki/Zachman_Framework
7. https://en.wikipedia.org/wiki/Soft_systems_methodology
8. https://en.wikipedia.org/wiki/Taboo

UNIT I

SYLLABUS

Overview of Free/ Open Source Software: The Open Source Definition - Examples of OSD Compliant Licenses - Examples of Open Source Software Product – The Open Source Software Development Process – A History of Open Source software: The Berkeley Software Distribution – The Free Software Foundation – Linux – Apache – Mozilla – Open Source Software.

OVERVIEW OF FREE OPEN SOURCE SOFTWARE

- Free and open-source software (FOSS) is software that can be classified as both free software and open-source software.
- That is, anyone is freely licensed to use, copy, study, and change the software in any way, and the source code is openly shared so that people are encouraged to voluntarily improve the design of the software.
- This is in contrast to proprietary software, where the software is under restrictive copyright and the source code is usually hidden from the users.

BENEFITS:

- The benefits of using FOSS can include decreased software costs, increased security and stability (especially in regard to malware), protecting privacy, and giving users more control over their own hardware.
- Free, open-source operating systems such as Linux and descendents of BSD are widely utilized today, powering millions of servers, desktops, smartphones (e.g. Android), and other devices.
- Free software licenses and open-source licenses are used by many software packages. The open-source software movement is an online social movement behind widespread production and adoption of FOSS.

OPEN SOURCE DEFINITION

Open source software is computer software that has a source code available to the general public for use as is or with modifications. This software typically does not require a license fee. There are open source software applications for a variety of different uses such as office automation, web design, content management, operating systems, and communications. 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. These criteria include the right to:

• Redistribute the software without restriction;

• Access the source code;

• Modify the source code; and

• Distribute the modified version of the software.

In contrast, creators of proprietary software usually do not make their source code available to others to modify. When considering the advantages of open source software you should consider the open source product itself. Open source products vary in quality. OSS software does not come with phone support or personalized e-mail support. However, there are commercial service providers who will provide support. If you need a lot of support, consider whether the overall costs of using an open source product will be higher than that of a proprietary product.

Nobody owns or controls the term "Open Source", as it was deemed too broad and descriptive to be a trademark under US Law. However, in general use, open source software is software distributed under terms that comply with the Open Source Definition (OSD). The OSD is a document maintained by the Open Source Initiative(OSI).

Furthermore, it is eligible to bear the OSI Certified certification mark (Perens, 1999; Open Source Initiative, 2001a). According to the OSI, the OSI certified certification mark "applies to software, not to licenses" (Open Source Initiative, 2001a). However, in practice, the OSD has been used mainly as a licensing standard, and the OSI maintains a list of OSD-compliant licenses. The majority of OSS products in circulation are self-certified (they are distributed under the terms of a previously approved license, and are thus implicitly trusted to implement it properly) and are not evaluated by the OSI on a product-byproduct basis. Developers may also, of course, submit a new license for OSI approval.

Either way, the OSI Certified mark is used by attaching one of two notices to the software product, namely, "This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative" or, simply, "OSI Certified Open Source Software" (Open Source Initiative, 2001a).

According to the Open Source Definition (Open Source Initiative, 2001b): Open Source doesn't just mean access to the source code. The distribution terms of open source software must comply with the following criteria:

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

2. Source Code: The program must include source code, and must follow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a

reasonable reproduction cost- preferably, downloading via the internet without charge. The source code must be the preferred form in which a programmer would modify the program.

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

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

5. No Discrimination against Persons or Groups: The license must not discriminate against any person or group of persons.

6. No Discrimination against Fields of Endeavor: The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

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

8. License must not be specific to a product: The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License must not contaminate other software: The license must not place restrictions on other software that is distributed along with the licensed software.

For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

EXAMPLES OF OSD-COMPLIANT LICENSES

The Open Source Initiative (2001c) provides a list of licenses that have been reviewed

and found to be compliant with the OSD. There are 21 licenses on the list, namely

The GNU General Public License (GPL)

The GNU Lesser Public License (LGPL)

The Berkeley Software Distribution (BSD) License

The MIT License

The Artistic License

The Mozilla Public License (MPL)

The Qt Public License (QPL)

The IBM Public License

The MITRE Collaborative Virtual Workspace License (CVW License)

The Ricoh Source Code Public License

The Python License

The zlib/libpng license

The Apache Software License

The Vovida Software License

The Sun Internet Standards Source License (SISSL)

The Intel Open Source License

The Jabber Open Source License

The Nokia Open Source License

The Sleepycat License

The Nethack General Public License

Since all of these licenses conform to the OSD, we will limit our comments to the more distinctive qualities of the most widely used licenses. The GPL and LGPL were created by Richard Stallman's Free Software Foundation (FSF) and, in fact, predate the coining of the term Open Source. There is an enormous amount of GPL-licenses software in circulation- 11723 independent projects hosted at the SourceForge website alone and the FSF itself has produced over 170 mature products, collectively referred to as the GNU Project.

SOME EXAMPLES OF OPEN SOURCE SOFTWARE

Accounting:

- SQL-Ledger (accounting system)

Anti-virus:

- ClamAV

Databases:

- LDAP
- MySQL (database)
- PostgreSQL (relational database with ability to do stored procedures)

Knowledge Management:

- Plone (open source content management system)
- Knowledge Tree

Domain Name Servers:

- Bind
- PowerDNS

Telephony:

- Asterisk (A Phone system [PBX] that also supports Voice Over IP technology)
- Elastix
- FreePBX
- Trixbox CE

E-mail Servers:

- PostFix
- QMail
- Sendmail

File Servers:

- FreeNAS
- OpenFiler
- Samba

Medical Software:

- http://www.oemr.org
- http://en.wikipedia.org/wiki/List_of_open_source_healthcare_software

Other Valuable Systems (servers & desktops):

- Apache (web server)
- CentOS (Linux distribution from Red Hat's development efforts)
- Fedora (Linux destop system)
- JBoss (J2EE server for Enterprise Java Development)
- Slackware (Linux distribution)
- Tomcat (Java servlet container)
- Ubuntu (a Linux desktop operating system)
- Zope (Content management system and portal)

Productivity Software:

- Evolution (calendar, contact manager and e-mail client)
- Firefox (web browser)
- Gimp (image manipulation program)
- Open Office (word processor, spreadsheet, etc.)
- Thunderbird (e-mail client, news aggregator, etc.)

Programming Languages:

C, C++, Mono, PHP, Python, Perl, Ruby, TcL

Spam Filtering:

- AmavisD
- PostGrey
- SpamAssign

Routing/Networking:

- DHCPD
- IPTables
- PF Sense

Virtualization:

- KVM
- Xen

OPEN SOURCE SOFTWARE DEVELOPMENT PROCESS

Open-source software development is the process by which open-source software, or similar software whose source code is publicly available, is developed. These are software products available with its source code under an open-source license to study, change, and improve its design. Examples of some popular open-source software products are Mozilla Firefox, Google Chromium, Android, LibreOffice and the VLC media player. Open-source software development has been a large part of the creation of the World Wide Web as we know it, with Tim Berners-Lee contributing his HTML code development as the original platform upon which the internet is now built.

In his 1997 essay *The Cathedral and the Bazaar*, open-source evangelist Eric S. Raymond suggests a model for developing OSS known as the *bazaar* model. Raymond likens the development of software by traditional methodologies to building a cathedral, "carefully crafted by individual wizards or small bands of mages working in splendid isolation". He suggests that all software should be developed using the bazaar style, which he described as "a great babbling bazaar of differing agendas and approaches

In the traditional model of development, which he called the *cathedral* model, development takes place in a centralized way. Roles are clearly defined. Roles include people dedicated to designing (the architects), people responsible for managing the project, and people responsible for implementation. Traditional software engineering follows the cathedral model.

The bazaar model, however, is different. In this model, roles are not clearly defined. Gregorio Robles suggests that software developed using the bazaar model should exhibit the following patterns:

Users should be treated as co-developers

The users are treated like co-developers and so they should have access to the source code of the software. Furthermore, users are encouraged to submit additions to the software, code fixes for the software, bug reports, documentation etc. Having more co-developers increases the rate at which the software evolves. Linus's law states, "Given enough eyeballs all bugs are shallow." This means that if many users view the source code, they will eventually find all bugs and suggest how to fix them. Note that some users have advanced programming skills, and furthermore, each user's machine provides an additional testing environment. This new testing environment offers that ability to find and fix a new bug.

Early releases

The first version of the software should be released as early as possible so as to increase one's chances of finding co-developers early.

Frequent integration

Code changes should be integrated (merged into a shared code base) as often as possible so as to avoid the overhead of fixing a large number of bugs at the end of the project life cycle. Some open source projects have nightly builds where integration is done automatically on a daily basis.

Several versions

There should be at least two versions of the software. There should be a buggier version with more features and a more stable version with fewer features. The buggy version (also called the development version) is for users who want the immediate use of the latest features, and are willing to accept the risk of using code that is not yet thoroughly tested. The users can then act as co-developers, reporting bugs and providing bug fixes.

High modularization

The general structure of the software should be modular allowing for parallel development on independent components.

Dynamic decision making structure

There is a need for a decision making structure, whether formal or informal, that makes strategic decisions depending on changing user requirements and other factors. Cf. Extreme programming.

Data suggests, however, that OSS is not quite as democratic as the bazaar model suggests. An analysis of five billion bytes of free/open source code by 31,999 developers shows that 74% of the code was written by the most active 10% of authors. The average number of authors involved in a project was 5.1, with the median at 2.

Open source software is usually easier to obtain than proprietary software, often resulting in increased use. Additionally, the availability of an open source implementation of a standard can increase adoption of that standard. It has also helped to build developer loyalty as developers feel empowered and have a sense of ownership of the end product.

Moreover, lower costs of marketing and logistical services are needed for OSS. OSS also helps companies keep abreast of technology developments. It is a good tool to promote a company's image, including its commercial products. The OSS development approach has helped produce reliable, high quality software quickly and inexpensively.

Open source development offers the potential for a more flexible technology and quicker innovation. It is said to be more reliable since it typically has thousands of independent programmers testing and fixing bugs of the software. Open source is not dependent on the company or author that originally created it. Even if the company fails, the code continues to exist and be developed by its users. Also, it uses open standards accessible to everyone; thus, it does not have the problem of incompatible formats that exist in proprietary software.

It is flexible because modular systems allow programmers to build custom interfaces, or add new abilities to it and it is innovative since open source programs are the product of collaboration among a large number of different programmers. The mix of divergent perspectives, corporate objectives, and personal goals speeds up innovation.

Moreover, free software can be developed in accord with purely technical requirements. It does not require thinking about commercial pressure that often degrades the quality of the software. Commercial pressures make traditional software developers pay more attention to customers' requirements than to security requirements, since such features are somewhat invisible to the customer.

It is sometimes said that the open source development process may not be well defined and the stages in the development process, such as system testing and documentation may be ignored. However this is only true for small (mostly single programmer) projects. Larger, successful projects do define and enforce at least some rules as they need them to make the teamwork possible. In the most complex projects these rules may be as strict as reviewing even minor change by two independent developers.

In terms of security, open source may allow hackers to know about the weaknesses or loopholes of the software more easily than closed-source software. It depends on control mechanisms in order to create effective performance of autonomous agents who participate in virtual organizations.

Development tools

In OSS development, tools are used to support the development of the product and the development process itself.

Revision control systems such as Concurrent Versions System (CVS) and later Subversion (SVN) and Git are examples of tools, often themselves open source, help manage the source code files and the changes to those files for a software project. The projects are frequently hosted and published on sites like Launchpad, Bitbucket, and GitHub.

Open source projects are often loosely organized with "little formalized process modeling or support", but utilities such as issue trackers are often used to organize open source software development. Commonly used bug trackers include Bugzilla and Redmine.

Tools such as mailing lists and IRC provide means of coordination among developers. Centralized code hosting sites also have social features that allow developers to communicate.

A HISTORY OF OPEN SOURCE SOFTWARE

In the 1950s and 1960s, computer operating software and compilers were delivered as a part of hardware purchases without separate fees. At the time, source code, the human-readable form of software, was generally distributed with the software providing the ability to fix bugs or add new functions. Universities were early adopters of computing technology. Many of the modifications developed by universities were openly shared, in keeping with the academic principles of sharing knowledge, and organizations sprung up to facilitate sharing. As large-scale operating systems matured, fewer organizations allowed modifications to the operating software, and eventually such operating systems were closed to modification. However, utilities and other added-function applications are still shared and new organizations have been formed to promote the sharing of software.

In 1969 the Advanced Research Projects Agency Network (ARPANET) was build, a transcontinental, high-speed computer network. The network (later succeeded by the Internet) simplified this exchange of software code and did further spread shared code and the hacking culture.

Some free software which was developed in the 1970s continues to be developed and used, such as TeX (developed by Donald Knuth) and SPICE.

Free software before the 1980s

In the 1950s and into the 1960s almost all software was produced by academics and corporate researchers working in collaboration, often shared as public domain software. As such, it was generally distributed under the principles of openness and co-operation long established in the fields of academia, and was not seen as a commodity in itself. Such communal behavior became later a central element of the so-called hacking culture (a term with a positive connotation among open source programmers). At this time, source code, the human-readable form of software, was generally distributed with the software machine code because users frequently modified the software themselves, because it would not run on different hardware or OS without modification, and also to fix bugs or add new functions.

In 1969 the Advanced Research Projects Agency Network (ARPANET) was build, a transcontinental, high-speed computer network. The network (later succeeded by the Internet)

simplified this exchange of software code and did further spread shared code and the hacking culture.

Some free software which was developed in the 1970s continues to be developed and used, such as TeX (developed by Donald Knuth) and SPICE.

Initial decline of free software

Software was not considered copyrightable before 1974 the US Commission on New Technological Uses of Copyrighted Works (CONTU) decided that "computer programs, to the extent that they embody an author's original creation, are proper subject matter of copyright", Therefore, software had no licenses attached and was shared as public domain software, typically with source code.

By the late 1960s change was coming: as operating systems and programming language compilers evolved, software production costs were dramatically increasing relative to hardware. In the *United States vs. IBM* antitrust suit, filed 17 January 1969, the U.S. government charged that bundled software was anticompetitive.[12] While some software continued to come at no cost, there was a growing amount of software that was for sale only under restrictive licences.

In the early 1970s AT&T distributed early versions of Unix at no cost to government and academic researchers, but these versions did not come with permission to redistribute or to distribute modified versions, and were thus not free software in the modern meaning of the phrase.

After Unix became more widespread in the early 1980s, AT&T stopped the free distribution and charged for system patches. As it is quite difficult to switch to another architecture, most researchers paid for a commercial licence.

In 1976 Bill Gates wrote an essay entitled Open Letter to Hobbyists, in which he expressed dismay at the widespread sharing of Microsoft's product Altair BASIC by hobbyists without paying its licensing fee.

In 1979, AT&T began to enforce its licences when the company decided it might profit by selling the Unix system.

In an announcement letter dated 8 February 1983 IBM inaugurated a policy of no longer distributing sources with purchased software.

SHARE program library

The SHARE users group, founded in 1955, began collecting and distributing free software. The first documented distribution from SHARE was dated 17 October 1955.

The "SHARE Program Library Agency" (SPLA) distributed information and software, notably on magnetic tape.

DECUS tapes

In the early 1980s, the so-called *DECUS tapes* were a worldwide system for transmission of free software for users of DEC equipment. Operating systems were usually proprietary software, but many tools like the *TECO* editor, *Runoff* text formatter, or *List* file listing utility, etc. were developed to make users' lives easier, and distributed on the DECUS tapes. The 1981 Decus tape was probably the most innovative by bringing the Lawrence Berkeley Laboratory *Software Tools Virtual Operating System* which permitted users to use a Unix-like system on DEC 16-bit PDP-11s and 32-bit VAXes running under the VMS operating system

Launch of the free software movement

In 1983, Richard Stallman launched the GNU Project to write a complete operating system free from constraints on use of its source code. Particular incidents that motivated this include a case where an annoying printer couldn't be fixed because the source code was withheld from users. In 1989, the first version of the GNU General Public License was published. A slightly updated version 2 was published in 1991. In 1989, some GNU developers formed the company Cygnus Solutions.

Linux (1991–)

The Linux kernel, started by Linus Torvalds, was released as freely modifiable source code in 1991. The license wasn't a free software license, but with version 0.12 in February 1992. Among Linux distributions, Debian GNU/Linux, begun by Ian Murdock in 1993, is noteworthy for being explicitly committed to the GNU and FSF principles of free software.

The Debian developers' principles are expressed in the Debian Social Contract. Since its inception, the Debian project has been closely linked with the FSF, and in fact was sponsored by the FSF for a year in 1994–1995.

In 1997, former Debian project leader Bruce Perens also helped found Software in the Public Interest, a non-profit funding and support organization for various free software projects.

Since 1996, the Linux kernel has included proprietary licensed components, so that it was no longer entirely free software. Therefore, the Free Software Foundation Latin America released in 2008 a modified version of the Linux-kernel called Linux-libre, where all proprietary and non-free components were removed.

The free BSDs (1993)

When the USL v. BSDi lawsuit was settled out of court in 1993, FreeBSD and NetBSD (both derived from 386BSD) were released as free software. In 1995, OpenBSD forked from NetBSD. In 2004, Dragonfly BSD forked from FreeBSD.

The X Window System was created in 1984, and became the de facto standard window system in desktop free software operating systems by the mid-1990s. In 2003, a proprietary Unix vendor and former Linux distribution vendor called SCO alleged that Unix intellectual property had been inappropriately copied into the Linux kernel, and sued IBM, claiming that it bore responsibility for this.

In 2004 the European Commission found Microsoft guilty of anti-competitive behavior with respect to interoperability in the workgroup software market. Microsoft had formerly settled United States v. Microsoft in 2001, in a case which charged that it illegally abused its monopoly power to force computer manufacturers to preinstall Internet Explorer.

In 2008 the International Organization for Standardization published Microsoft's Office Open XML as an international standard, which crucially meant that it, and therefore Microsoft Office, could be used in projects where the use of open standards were mandated by law or by policy. As of 2012, no fully correct open source implementation of OOXML exists, which validates the critics' remarks about OOXML being difficult to implement and underspecified. In September 2008, Google released the first version of Android, a new smartphone operating system, as open source (some Google applications that are sometimes but not always bundled with Android are not open source).

Chromium OS (2009–)

Until recently, Linux was still a relatively uncommon choice of operating system for desktops and laptops. However, Google's Chrome books, running Chrome OS which is essentially a web thin client, have captured 20-25% of the sub-$300 US laptop market. Chrome OS is built from the open source Chromium OS, which is based on Linux, in much the same way that versions of Android shipped on commercially available phones are built from the open source version of Android.

BERKELEY SOFTWARE DISTRIBUTION

Berkeley Software Distribution (BSD) was a Unix operating system derivative developed and distributed by the Computer Systems Research Group (CSRG) of the University of California, Berkeley, from 1977 to 1995. Today the term "BSD" is often used non-specifically to refer to any of the BSD descendants which together form a branch of the family of Unix-like operating systems. Operating systems derived from the original BSD code remain actively developed and widely used.

Short for Berkeley Software Distribution, BSD is a Unix-like operating system first introduced in late 1977. Originally titled 1BSD, it was developed at the Computer System Research Group (CSRG) of the University of California at Berkeley. Today, BSD comes in various flavors such as BSDi Internet Server (BSD/OS), FreeBSD, NetBSD, and OpenBSD below is a brief introduction to each of these flavors of BSD.

BSDi Internet Server (BSD/OS)

BSDi or BSD Inc. was founded in 1991 by some of the leading CSRG computer scientists. BSD/OS is a full-function, POSIX-compatible, Unix-like operating system for the 386, 486, and Pentium architectures. BSDI believes in one-stop shopping, high levels of integration and a product that requires payment of no external licensing fees.

FreeBSD

Developed and maintained by a large team of individuals. FreeBSD is a full function, POSIX-compatible, Unix-like operating system for Intel compatible (x86), DEC Alpha and PC-98 architectures.

NetBSD

Developed and maintained by a large team of individuals. NetBSD is another free version of BSD compatible with a very large variety of platforms, from 64-bit Alpha servers to handheld devices.

OpenBSD

Developed and maintained by a large team of individuals. OpenBSD is multi-platform 4.4BSD-based Unix-like operating system. macOS X.Apple operating system based on BSD.

Relationship to Research Unix

Starting with the 8th Edition, versions of Research Unix at Bell Labs had a close relationship to BSD. This began when 4.1cBSD for the VAX was used as the basis for Research Unix 8th Edition. This continued in subsequent versions, such as the 9th Edition, which incorporated source code and improvements from 4.3BSD. The result was that these later versions of Research Unix were closer to BSD than they were to System V. In a Usenet posting from 2000, Dennis Ritchie described this relationship between BSD and Research Unix.

Relationship to System V

Eric S. Raymond summarizes the longstanding relationship between System V and BSD, stating, "The divide was roughly between longhairs and shorthairs; programmers and technical people tended to line up with Berkeley and BSD, more business-oriented types with AT&T and System V.

In 1989, David A. Curry wrote about the differences between BSD and System V. He characterized System V as being often regarded as the "standard Unix." However, he described BSD as more popular among university and government computer centers, due to its advanced features and performance.

Berkeley sockets

Berkeley's Unix was the first Unix to include libraries supporting the Internet Protocol stacks: Berkeley sockets. A Unix implementation of IP's predecessor, the ARPAnet's NCP, with FTP and Telnet clients, had been produced at U. Illinois in 1975, and was available at Berkeley. However, the memory scarcity on the PDP-11 forced a complicated design and performance problems.

Binary compatibility

BSD operating systems can run much native software of several other operating systems on the same architecture, using a binary compatibility layer. Much simpler and faster than emulation, this allows, for instance, applications intended for Linux to be run at effectively full speed. This makes BSDs not only suitable for server environments, but also for workstation ones, given the increasing availability of commercial or closed-source software for Linux only. This also allows administrators to migrate legacy commercial applications, which may have only supported commercial Unix variants, to a more modern operating system, retaining the functionality of such applications until they can be replaced by a better alternative.

Standards adherence

Current BSD operating system variants support many of the common IEEE, ANSI, ISO, and POSIX standards, while retaining most of the traditional BSD behavior. Like AT&T Unix, the BSD kernel is monolithic, meaning that device drivers in the kernel run in privileged mode, as part of the core of the operating system.

THE FREE SOFTWARE FOUNDATION

- "Free software is a matter of liberty, not price.
- To understand the concept, you should think of free as in **free speech (right)**, not as in **free beer** (gift).
- Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software.
  - The freedom to run the program, for any purpose (freedom 0).
  - The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
  - The freedom to redistribute copies so you can help your neighbor (freedom 2).

The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this."

"A program is free software if users have all of these freedoms. Thus, you should be 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.

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."

Very counter-culture

Hacker is considered a "good-guy"

"Hacker (computer security) someone involved in computer security/insecurity

Hacker (programmer subculture), a programmer subculture originating in the US academia in the 1960s, which is nowadays mainly notable for the free software/open source movement

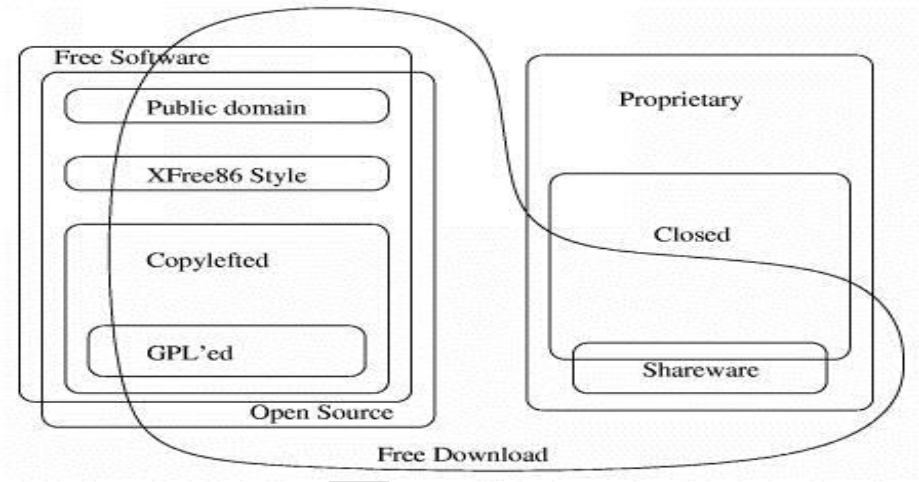
Hacker (hobbyist), an enthusiastic home computer hobbyist" http://en.wikipedia.org/wiki/Hacker

Cracker is a "bad-guy"

A cracker is someone who cracks software or digital media

"Software cracking is the modification of software to remove protection methods: copy protections, trial/demo version, serial number, hardware key, date checks, CD check or software annoyances like nag screens and adware".

- General Public License - GPL in 1991
  - The community rather than the company
  - Copyleft
  - No limits on software released under this license
- Opposite of proprietary software



LINUX

Linux is the best-known and most-used open source operating system. As an operating system, Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware.

For the purposes of this page, we use the term "Linux" to refer to the Linux kernel, but also the set of programs, tools, and services that are typically bundled together with the Linux kernel to provide all of the necessary components of a fully functional operating system. Some people, particularly members of the Free Software Foundation, refer to this collection as GNU/Linux, because many of the tools included are GNU components. However, not all Linux installations use GNU components as a part of their operating system. Android, for example, uses a Linux kernel but relies very little on GNU tools.

How does Linux differ from other operating systems?

In many ways, Linux is similar to other operating systems you may have used before, such as Windows, OS X, or iOS. Like other operating systems, Linux has a graphical interface, and types of software you are accustomed to using on other operating systems, such as word processing applications, have Linux equivalents. In many cases, the software's creator may have made a Linux version of the same program you use on other systems. If you can use a computer or other electronic device, you can use Linux.

But Linux also is different from other operating systems in many important ways. First, and perhaps most importantly, Linux is open source software. The code used to create Linux is free and available to the public to view, edit, and—for users with the appropriate skills—to contribute to.

Linux is also different in that, although the core pieces of the Linux operating system are generally common, there are many distributions of Linux, which include different software options. This means that Linux is incredibly customizable, because not just applications, such as word processors and web browsers, can be swapped out. Linux users also can choose core components, such as which system displays graphics, and other user-interface components.

What is the difference between Unix and Linux?

We may have heard of Unix, which is an operating system developed in the 1970s at Bell Labs by Ken Thompson, Dennis Ritchie, and others. Unix and Linux are similar in many ways, and in fact, Linux was originally created to be similar to Unix. Both have similar tools for interfacing with the systems, programming tools, filesystem layouts, and other key components. However, Unix is not free. Over the years, a number of different operating systems have been created that attempted to be "unix-like" or "unix-compatible," but Linux has been the most successful, far surpassing its predecessors in popularity.

Who uses Linux?

We are probably already using Linux, whether you know it or not. Depending on which user survey you look at, between one- and two-thirds of the webpages on the Internet are generated by servers running Linux.

Companies and individuals choose Linux for their servers because it is secure, and you can receive excellent support from a large community of users, in addition to companies like Canonical, SUSE, and Red Hat, which offer commercial support.

Many of the devices you own probably, such as Android phones, digital storage devices, personal video recorders, cameras, wearables, and more, also run Linux. Even your car has Linux running under the hood.

Who "owns" Linux?

By virtue of its open source licensing, Linux is freely available to anyone. However, the trademark on the name "Linux" rests with its creator, Linus Torvalds. The source code for Linux is under copyright by its many individual authors, and licensed under the GPLv2 license. Because Linux has such a large number of contributors from across multiple decades of development, contacting each individual author and getting them to agree to a new license is

virtually impossible, so that Linux remaining licensed under the GPLv2 in perpetuity is all but assured.

How was Linux created?

Linux was created in 1991 by Linus Torvalds, a then-student at the University of Helsinki. Torvalds built Linux as a free and open source alternative to Minix, another Unix clone that was predominantly used in academic settings. He originally intended to name it "Freax," but the administrator of the server Torvalds used to distribute the original code named his directory "Linux" after a combination of Torvalds' first name and the word Unix, and the name stuck.

Most of the Linux kernel is written in the C programming language, with a little bit of assembly and other languages sprinkled in. If you're interested in writing code for the Linux kernel itself, a good place to get started is in the Kernel Newbies FAQ, which will explain some of the concepts and processes you'll want to be familiar with.

But the Linux community is much more than the kernel, and needs contributions from lots of other people besides programmers. Every distribution contains hundreds or thousands of programs that can be distributed along with it, and each of these programs, as well as the distribution itself, need a variety of people and skill sets to make them successful, including:

- Testers to make sure everything works on different configurations of hardware and software, and to report the bugs when it does not.
- Designers to create user interfaces and graphics distributed with various programs.
- Writers who can create documentation, how-tos, and other important text distributed with software.
- Translators to take programs and documentation from their native languages and make them accessible to people around the world.
- Packagers to take software programs and put all the parts together to make sure they run flawlessly in different distributions.

- Evangelists to spread the word about Linux and open source in general.
- And of course developers to write the software itself.

The GNU/Linux distributions that are entirely free as in freedom. All of the distributions that follow are installable to a computer's hard drive; most can be run live.

The Free Software Foundation recommends and endorses these GNU/Linux distros, although we do not try to judge or compare them based on any criterion other than freedom; therefore, we list them in alphabetical order. We encourage you to read these brief descriptions and to consult their respective web sites and other information to choose the one best for you.

These distros are ready-to-use full systems whose developers have made a commitment to follow the Guidelines for Free System Distributions. This means these distros will include, and propose, exclusively free software. They will reject nonfree applications, nonfree programming platforms, nonfree drivers, nonfree firmware "blobs", nonfree games, and any other nonfree software, as well as nonfree manuals or documentation.

If one of these distros ever does include or propose anything nonfree, that must have happened by mistake, and the developers are committed to removing it. If you find nonfree software or documentation in one of these distributions, you can report the problem, and earn GNU Bucks, while we inform the developers so they can fix the problem.

Fixing freedom bugs is an ethical requirement for listing a distro here; therefore, we list only distros with a development team that has told us it will remove any nonfree software that might be found in them. Usually the team consists of volunteers, and they don't make legally binding commitments to users; but if we find out a distro is not properly maintained, we will de-list it.

We hope the other existing GNU/Linux distributions will become entirely free software so that we can list them here. If you wish to improve the state of free distros, helping to develop an existing free distro contributes more than starting a new one.

All of the distributions that follow are installable to a computer's hard drive; most can be run live. Not all hardware works in the free world; each distro's site should say which hardware it supports.

APACHE

The **Apache Software Foundation** (**ASF**) is an American non-profit corporation (classified as 501(c)(3) in the United States) to support Apache software projects, including the Apache HTTP Server. The ASF was formed from the Apache Group and incorporated in Delaware, U.S., in June 1999.

The Apache Software Foundation is a decentralized open source community of developers. The software they produce is distributed under the terms of theApache License and is free and open-source software (FOSS). The Apache projects are characterized by a collaborative, consensus-based development process and an open and pragmatic software license.

Each project is managed by a self-selected team of technical experts who are active contributors to the project. The ASF is a meritocracy, implying that membership of the foundation is granted only to volunteers who have actively contributed to Apache projects. The ASF is considered a second generation open-source organization, in that commercial support is provided without the risk of platform lock-in.

Among the ASF's objectives are: to provide legal protection[4] to volunteers working on Apache projects; to prevent the *Apache* brand name from being used by other organizations without permission.

The ASF also holds several ApacheCon conferences each year, highlighting Apache projects and related technology.

The history of the Apache Software Foundation is linked to the Apache HTTP Server, development beginning in February 1993. A group of eight developers started working on enhancing the NCSA HTTPd daemon. They came to be known as the Apache Group. On March 25, 1999, the Apache Software Foundation was formed.The first official meeting of the Apache Software Foundation was held on April 13, 1999, and by general consent that the initial membership list of the Apache Software Foundation, would be: Brian Behlendorf, Ken Coar, Miguel Gonzales, Mark Cox, Lars Eilebrecht, Ralf S. Engelschall, Roy T. Fielding, Dean Gaudet, Ben Hyde, Jim Jagielski, Alexei Kosut, Martin Kraemer, Ben Laurie, Doug MacEachern, Aram Mirzadeh, Sameer Parekh, Cliff Skolnick, Marc Slemko, William (Bill) Stoddard, Paul Sutton, Randy Terbush and Dirk-Willem van Gulik. After a series of additional meetings to elect board members and resolve other legal matters regarding incorporation, the effective incorporation date of the Apache Software Foundation was set to June 1, 1999.

The name 'Apache' was chosen from respect for the Native American Apache Nation, well known for their superior skills in warfare strategy and their inexhaustible endurance. It also makes a pun on "a patchy web server"—a server made from a series of patches—but this was not its origin. The group of developers who released this new software soon started to call themselves the "Apache Group".

Oracle, IBM, and the Apache Software Foundation jointly announced last week that OpenOffice.org would become an official Apache project. OpenOffice.org is an important piece of free software, and many of its supporters suggest that this change will give them more control over the project's future direction. However, users and contributors should be aware that, as part of this transition, it will become easier for proprietary software developers to distribute OpenOffice.org as nonfree software.

All Apache projects are distributed under the terms of the Apache License. This is a non-copyleft free software license; anybody who receives the software can distribute it to others under nonfree terms. Such a licensing strategy represents a significant policy change for OpenOffice.org. Previously, the software was distributed under the terms of the GNU Lesser General Public License (LGPL). The LGPL is a weak copyleft license, so programs that merely link to the software can be released under nonfree terms, but the software covered by the LGPL must always be released, along with its source code, under the LGPL's terms. Free software developers are clearly comfortable with a partial copyleft when it's appropriate; in numerous surveys of free software projects, the LGPL is commonly listed as the second-most popular license (after the GNU General Public License), or else follows close behind.

While we do recommend the Apache License in specific situations, we do not believe it is the best choice for software like OpenOffice.org. This situation calls for copyleft, because the gains free software stands to make from a non-copyleft license don't justify giving a handout to proprietary software developers.

Fortunately, there's a ready alternative for people who want to work with a productivity suite that does more to protect their freedom: LibreOffice. Anybody who's comfortable with OpenOffice.org will find a familiar interface and feature set in LibreOffice, because it was originally based on the same source code. Since September 2010, numerous contributors have been working to improve the software, and the project's legal steward, The Document Foundation, is committed to keeping it licensed under the LGPL.

LibreOffice's commitment to user freedom does not end at the license of its source code. Like OpenOffice.org, the software's built-in extension manager makes it easy to add new features, but unlike OpenOffice.org, its extension database only lists add-ons that are under a free license. OpenOffice.org points to a database that includes proprietary extensions, and doesn't always provide clear licensing information. This approach to extensions risks turning free software into a platform for the development and promotion of proprietary extras.

Anybody who plans to use or contribute to one of these productivity suites should understand how these policies affect them, and consider which better complement their own goals. While both pass the most important test of being free software, we recommend LibreOffice because its policies do significantly more to promote the cause of free software.

MOZILLA

Mozilla is a free-software community created in 1998 by members of Netscape. The Mozilla community uses, develops, spreads and supports Mozilla products, thereby promoting exclusively free software and open standards, with only minor exceptions.The community is supported institutionally by the Mozilla Foundation and its tax-paying subsidiary, the Mozilla Corporation.

Mozilla's products include the Firefox web browser, Thunderbird e-mail client, Firefox OS mobile operating system, Bugzilla bug tracking system,Gecko layout engine and others. During 2017, Mozilla acquired Pocket, a "read-it-later-online" service.

Firefox is a web browser, and is Mozilla's flagship software product. It is available in both desktop and mobile versions. Firefox uses the Gecko layout engine to render web pages, which implements current and anticipated web standards. As of late 2015, Firefox had approximately 10-11% of worldwide usage share of web browsers, making it the 4th most-used web browser.

Firefox began as an experimental branch of the Mozilla codebase by Dave Hyatt, Joe Hewitt and Blake Ross. They believed the commercial requirements of Netscape's sponsorship and developer-driven feature creep compromised the utility of the Mozilla browser.[48] To combat what they saw as the Mozilla Suite's software bloat, they created a stand-alone browser, with which they intended to replace the Mozilla Suite.

Firefox was originally named *Phoenix* but the name was changed so as to avoid trademark conflicts with Phoenix Technologies. The initially-announced replacement, *Firebird*,

provoked objections from the Firebird project community. The current name, Firefox, was chosen on February 9, 2004.

Firefox OS

Firefox OS was an open source operating system in development by Mozilla that aims to support HTML5 apps written using "open Web" technologies rather than platform-specific native APIs. The concept behind Firefox OS is that all user-accessible software will be HTML5 applications, that use Open Web APIs to access the phone's hardware directly via JavaScript.

Some devices using this OS include Alcatel One Touch Fire, ZTE Open, and LG Fireweb.

Thunderbird

Thunderbird is a free, open source, cross-platform email and news client developed by the volunteers of the Mozilla Community.

On July 16, 2012, Mitchell Baker announced that Mozilla's leadership had come to the conclusion that on-going stability was the most important thing for Thunderbird and that innovation in Thunderbird was no longer a priority for Mozilla. In that update Baker also suggested that Mozilla had provided a pathway for community to innovate around Thunderbird if the community chooses.

SeaMonkey

SeaMonkey (formerly the Mozilla Application Suite) is a free and open source cross platform suite of Internet software components including a web browser component, a client for sending and receiving email and Usenet newsgroup messages, an HTML editor (Mozilla Composer) and the ChatZilla IRC client.

On March 10, 2005, the Mozilla Foundation announced that it would not release any official versions of Mozilla Application Suite beyond 1.7.x, since it had now focused on the standalone applications Firefox and Thunderbird. SeaMonkey is now maintained by the SeaMonkey Council, which has trademarked the SeaMonkey name with help from the Mozilla Foundation.The Mozilla Foundation provides project hosting for the SeaMonkey developers.

Bugzilla

Bugzilla is a web-based general-purpose bug tracking system, which was released as open source software by Netscape Communications in 1998 along with the rest of the Mozilla codebase, and is currently stewarded by Mozilla. It has been adopted by a variety of organizations for use as a bug tracking system for both free and open source

software and proprietary projects and products, including the Mozilla Foundation, the Linux kernel, GNOME, KDE, Red Hat, Novell, Eclipse andLibreOffice.

Mozilla is open source and free software – any person or company is free to:

- run the program, for any purpose;
- study how the program works, and adapt it to their needs;
- redistribute copies at will;
- improve the program, and distribute the altered version.

All the source code for Mozilla is available under the Mozilla and Netscape Public Licenses, which are accepted as free software licenses by the Free Software Foundation.

The spirit of the MPL is that you are free to use Mozilla code in your applications and products – including proprietary products – provided that you make available any modifications you make to the actual Mozilla code base itself.

With free software, your business is not locked into the products of one company. You are free to control your own future.

OPEN SOURCE SOFTWARE

- In the beginning, all software was free
  - in the 1960s ,when IBM and others sold the first large-scale computers, these machines came with software which was free.
  - This software could be freely shared among users,
    - The software came written in a programming language, and it could be improved and modified.
- Then proprietary software dominated the software landscape
  - IBM and others realized that most users couldn't or didn't want to "fix" their own software and
  - There was money to be made in leasing software
- In mid-1970s it software was proprietary
  - users were not allowed to redistribute it,
  - that source code was not available
  - users could not modify the programs.
- In late 1970s and early 1980s, two different groups started what became known as the open source software movement:
- East coast, Richard Stallman, formerly a programmer at the MIT AI Lab, launched the GNU Project and the Free Software Foundation.
  - ultimate goal of the GNU Project was to build a free operating system

  - the GNU General Public License (GPL) was designed to ensure that the software produced by GNU will remain free, and to promote the production of more and more free software.
- West coast, the Computer Science Research Group (CSRG) of the University of California at Berkeley was improving the Unix system, and building applications which quickly become ``BSD Unix".
  - efforts were funded mainly by DARPA contracts
  - a network of Unix programmers around the world helped to debug, maintain and improve the system.
  - in late 1980s, distributed under the ``BSD license" (one of the first open source licenses).
  - Unfortunately, still contained some components that were proprietary requiring a license from AT&T
- During the 1980s and early 1990s, open source software continued its development, initially in several relatively isolated groups.
- Slowly, much of the software was integrated
- The various groups merged
- As a result of this i, complete operating environments could be built on top of Unix using open source software.
- Many Internet ISPs use UNIX as their operating system of choice.
- 1991-1992, the open source world improved
- In California, Bill Jolitz implementing a version of BSD Unix free of AT & T's copyright.
  - The work was covered by the BSD license making it completely free.
  - It included other free software GNU licenses
- Also during 1991-1992
- In Finland, Linus Torvalds, a Finnish computer science student, was implementing the first versions of Linux.
- Other people joined to collaboration to create the GNU/Linux operating system.
- By 1993, both GNU/Linux and BSD Unix were free stable operating environments.
  - Both continue to evolve
- "Open source is a development method for software that harnesses the power of distributed peer review and transparency of process.
- The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.
- The Open Source Initiative (OSI) is a non-profit corporation formed to educate about and advocate for the benefits of open source."
- OSI includes a standards body, maintaining the Open Source Definition for the good of the community.
- Today there are many who believe proprietary software is the only possible model
  - Microsoft.
- Recently the software industry has begun to considered free software as an option again.
  - Apple's OS X and Leopard are based on Unix
  - Google's Chrome

– Mozilla Firefox

Revolution OS

- "Revolution OS is a 2001 documentary which traces the history of GNU, Linux, and the open source and free software movements. "

UNIT I

POSSIBLE QUESTIONS

(8 MARKS)

1. List out the examples of OSD complaint licenses
2. List out the examples of open source software.
3. Describe about Free Software Foundation
4. Describe about Berkeley Software Distribution
5. Write a short note on
   i. i)Linux
   ii. ii)Apache
   iii. iii)Mozilla
6. Describe OSS with its criteria and list out the examples of open source software.
7. Describe the history of open source software.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-641021
Department of Computer Science
III B.Sc(CS) (BATCH 2015-2018)
Open Source Software
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS
ONLINE EXAMINATIONS **ONE MARK QUESTIONS**

S.No	Question	Option1	Option2	Option3	Option4			Answer
1	OSS stands for _____	Open Source Software	Open Standard Software	Open Standard Service	Open Source Service			Open Source Software
2	In _____ the source code used in the software is available to anyone to examine, evaluate and adapt.	close source	open source	shared source	proprietary source			open source
3	Which of the following is not an example of open source software?	Linux	Mozilla	Windowss	Apache			Windows
4	The human readable code of the program is known as _____	source code	byte code	machine code	object code			source code
5	In _____ the source code used in the software is available to anyone to examine, evaluate and adapt.	close source	shared source	open source	proprietary source			open source
6	The human readable code of the program is known as _____	source code	byte code	machine code	object code			source code
7	_____ is a proprietary software made available free of charge	Free software	Open source	Freeware	Malware			Freeware
8	BSD stands for _____	Berkeley Software Distribution	Berkeley Source Distribution	Berkeley Software Destruction	Berkeley Source Destruction			Berkeley Software Distribution

9	Wikipedia is an example of_____ software	freeware	open source	proprietary	free			open source
10	Free software is also known as_____	open libre	software l	free libre	source libre			software libr
11	_____ is a matter of liberty, not price	free software	freeware	close software	open source			free software
12	MySQL is a _____ software	close source	open source	proprietary	free			open source
13	GDB stands for _____	GNU Data	GNU Dat	GNU Docu	GNU Debugger			GNU Debugger
14	BSD licenses also referred as_____	BSD style	BSD unig	BSD source	BSD type			BSD style
15	Proprietary software is also known as _____	close sour	open sour	shared sour	free source			close source
16	In open source software the licensor distributes the _____	byte cod	source co	machine co	object code			source code
17	In _____ licensor distributes the object code.	close sour	open sour	shared sour	free source			close source
18	OSI stands for _____	Open Standard Initiative	Open Source Initiative	Open Standard Instruction	Open Source Instruction			Open Source Initiative
19	Sublicensing is prohibited in _____software.	freeware	open source	free	proprietary			proprietary
20	FSF stands for_____	Free Software Foundation	Free System Foundation	Free Service Foundation	Free Source Foundation			Free Software Foundation
21	Which of the following license the software cannot be mixed with non-free software	MIT	LGPL	GPL	BSD			GPL
22	_____ is a form of intellectual property , applicable to certain forms creative works	License	patent	copyleft	copyright			copyright
23	The term free in free software represents _____	empty	freedom	freeware	free cost			freedom

24	Berkeley Software Distribution is sometimes called _____	Berkeley Unix	Berkeley Linux	Berkeley Kernel	Berkeley Domain			Berkeley Unix
25	Life cycle paradigm of software engineering is called the _____ model.	waterfall	spiral	agile	spiral			waterfall
26	The term _____ also refers to the document that specifically describes these permissions and rights.	License	patent	copyleft	copyright			License
27	Which of the following license that can be relicensed by anyone?	E	M	public domain	LGPL			public domain
28	The reversed c in a full circle is the _____ symbol	copyright	license	patent	copyleft			copyleft
29	_____ license does not allow you to take modifications private.	MPL	LGPL	NPL	MIT			LGPL
30	_____ who participate frequently in newsgroups and discussions, but do not do any coding.	Project leaders	Maintainers	Posters	Occasional developers			Posters
31	Eric. Raymond wrote the Cathedral and the Bazaar model in _____	1998	1995	1994	1997			1997
32	_____ is the free software project funded by Netscape to build a WWW browser.	Mozilla	Internet Explorer	Chrome	Opera			Mozilla
33	_____ model adds an element of risk analysis to the software development process.	Scrum	Water fall	Spiral	Prototype			Spiral
34	_____ activity is the transition of the design specification into a software program.	Analysis	Coding	Testing	Maintenance			Coding

35	The principle of maintaining well defined boundaries between components is called _____	availability	modularity	c) Extension	subset			modularity
36	_____ is an interpreted language with lots of libraries	Java	Perl	C	C++			Perl
37	GCC stands for _____	GNU Complete Collection	GNU Complex Collection	GNU Compound Collection	GNU Compiler Collection			GNU Compiler Collection
38	The GNU project is free software, mass collaboration project, announced in 1983 by_____	Richard Stallman	Ken Thompson	Linus Torvalds	Bill Jolitz			Richard Stallman
39	Which of the following is widely used as a WWW server?	BIND name server	Send mail transport	Apache Web server	Samba file server			Apache Web server
40	L in the LGPL represents____	Local	Lesser	Legal Limit				Lesser
41	TCL stands for _____	Tools Computer Language	Tools Command Loader	Tools Command Language	Tools Command Linker			Tools Command Language
42	The c in a full circle is the _____ symbol	copyright	license	patent	copyleft			copyright
43	In _____ fees are for the software license, maintenance and upgrades.	free source	open source	shared source	close source			close source
44	In proprietary source, all upgrades, support and development are done by a _____	licensee	licensor	User	third parties			licensor
45	In _____ the licensee may do its own development and support or hire any third party to do it.	free source	open source	shared source	proprietary source			proprietary source

46	SDLC stands for _____	Software Development Life Cycle	Source Development Life Cycle	Software Deployment Life Cycle	Source Deployment Life Cycle			Software Development Life Cycle
47	Openoffice.org was established by _____	IBM	Microsoft	Sun Microsystems	Corel			Sun Microsystems
48	_____ is a non-profit corporation dedicated to managing and promoting the open source.	FSF	OSI	DARPA	ASF			OSI
49	_____ license contains special privileges for original copyright holder over your modifications	Public Domain	LGPL	NPL	GPL			NPL
50	_____ is a non-profit corporation to support Apache software project	FSF	OSI	DARPA	ASF			ASF
51	Who have the overall responsibility for the open source software development?	project leader	Volunteer developers	everyday users	posters			project leader
52	_____ is a bug tracker written in PERL language.	Bugzilla	Request tracker	Mantis	Trac			Request tracker
53	_____ is a web-based PHP/MySQL bug tracker.	Trac	Bugzilla	Request tracker	Mantis			Mantis
54	Which of the following is sophisticated bug tracker from the Mozilla house?	Bugzilla	Mantis	GNATS	LibreSource			Bugzilla
55	XP stands for_____	External Programming	Extreme Programming	Extensible Programming	Executable Programming			Extreme Programming

56	Which phase of the Spiral model determine objectives and constraints of the project and define the alternatives?	Risk analysis	Planning	Engineering	Customer Evolution			Planning
57	_____ helps to manage the files and codes of a project when several people are working on the project at the same time.	CVS	RPM	APT	SVN			CVS
58	_____ is a computer program that is used to debug other programs.	Compiler	Interpreter	Debugger	Loader			Debugger
59	LibreSource is a type of _____	debugger	compiler	interpreter	bugtracker			bugtracker

e

!

!

UNIT II

SYLLABUS

Qualification: Defining Open Source Software – Categorizing Defining Open Source Software – Specific Characteristics of Open Source Software Transformation: The OSS Development Process – Taboos and Norms in OSS Development – The OSS Development Life Cycle – Deriving a Framework for Analyzing OSS – Zachman‟s Framework for IS Architecture – CATWOE and Soft System Method – Deriving the Analytical Framework for OSS.

QUALIFICATION AND CATEGORIZING: DEFINING OPEN SOURCE SOFTWARE

The Qualification and Selection of Open Source software (QSOS) is a methodology for assessing Free/Libre Open Source Software. This methodology is released under the GFDL license. Several methods have been created to define an assessment process for free/open-source software. Some focus on some aspects like the maturity, the durability and the strategy of the organization around the open-source project itself. Other methodologies add functional aspects to the assessment process.

Existing methodologies

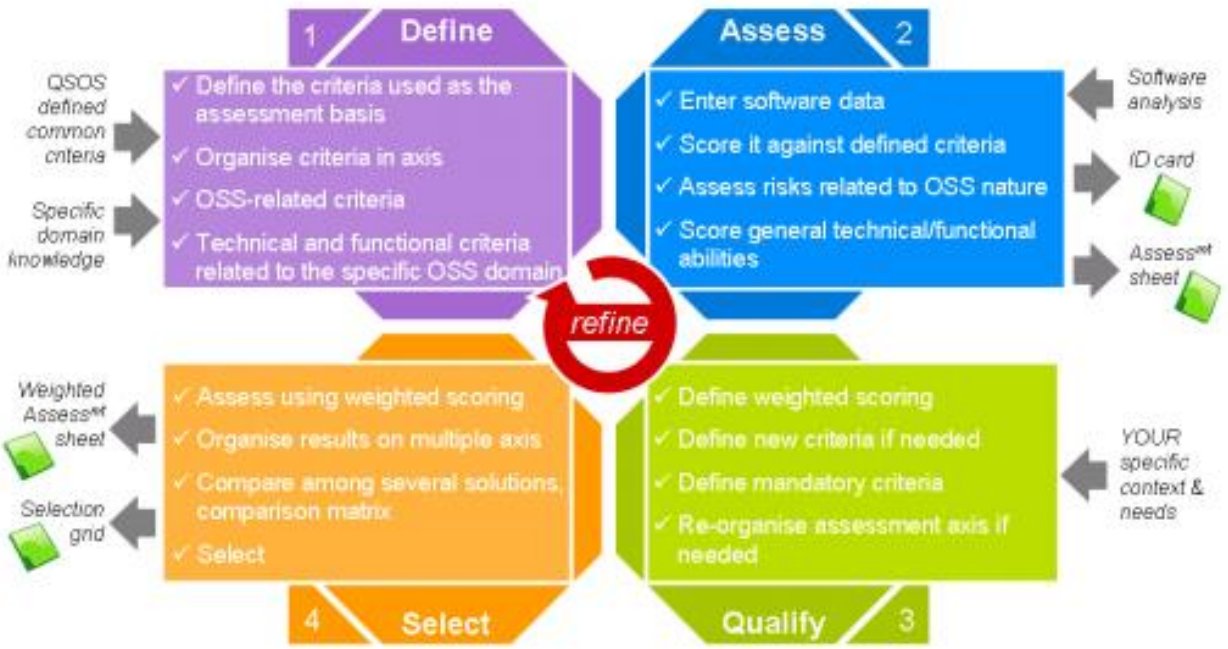
There are more than 20 different OSS evaluation methods.

- Open Source Maturity Model (OSMM) from Capgemini
- Open Source Maturity Model (OSMM) from Navica
- Open Source Maturity Model (OSSMM) by Woods and Guliani
- Methodology of Qualification and Selection of Open Source software (QSOS)
- Open Business Readiness Rating (OpenBRR)
- Open Business Quality Rating (OpenBQR)
- QualiPSo
- QualiPSo Model for Open Source Software Trustworthiness (MOSST)
- Towards A Trustworthiness Model For Open Source Software: How to evaluate Open Source Software
- QualOSS – Quality of Open Source
- Evaluation Framework for Open Source Software
- A Quality Model for OSS Selection
- Atos Origin Method for Qualification and Selection of Open Source Software (QSOS)

- Observatory for Innovation and Technological transfer on Open Source software (OITOS)
- Framework for OS Critical Systems Evaluation (FOCSE)

General approach

QSOS defines 4 steps that are part of an iterative process:



- 1 - Define and organise what will be assessed (common Open Source criteria and risks and technical domain specific functionalities),
- 2 - Assess the competing software against the criteria defined above and score these criteria individually,
- 3 - Qualify your evaluation by organising criteria into evaluation axes, and defining filtering (weightings, etc.) related to your context,
- 4 - Select the appropriate OSS by scoring all competing software using the filtering system designed in step 3.

Output documents

This process generates software assessing sheets as well as comparison grids. These comparison grids eventually assist the user to choose the right software depending on the context. These documents are also released under the free GNU FDL License. This allows them to be reused

and improved as well as to remain more objective. Assessment sheets are stored using an XML-based format.

Tools

Several tools distributed under the GPL license are provided to help users manipulate QSOS documents:

- Template editor: QSOS XUL Template Editor
- Assessment sheets editors:
  - QSOS XUL Editor
  - QSOS Qt Editor
  - QSOS Java Editor (under development)

SPECIFIC CHARACTERISTICS OF OPEN SOURCE SOFTWARE TRANSFORMATION

The development of open source software consists of planning, analysis, design, and implementation phases as in any other software model. However, there are unique characteristics of FOSS. In this section, we describe the main characteristic of Free and Open Source Software. In a typical FOSS, initially an individual or few volunteers involve in the project. Once the project is debut and successful then a community of project is established. Later other members from the community contribute to the project.

The Concurrent Versions System (CVS) helps is distributed development of FOSS. CVS is a client-server software revision control system. CVS keeps track of all changes in a set of files, and allows several developers to collaborate. CVS itself is a Free and Open Source Software. Globally distributed software development by virtual teams promises the flexibility, responsiveness, lower costs, and improved resource utilization. Modular Design In modular design software architecture is divided into components called modules.

Modular design supports abstraction, increased understanding of the system and concurrent development. Due to distributed nature of FOSS, its design must be modular that can easily incorporate into the main system. Modularity is favorable characteristics for open source production. Modular design with well-defined interfaces helps in effective collaborative development of FOSS. Figure 1 shows the modular design approach of FOSS.

Reusability

Reusability means segment of source code that can be used again to add new functionalities with little or no modification. This fits very well the characteristics of the Open Source production

process.FOSS licenses grants the rights to the developer to obtain the source code, inspect it, modify it, and distribute it. This mean FOSS licenses inherently encourages a developer to reuse

code. The reuse of code can be either within the project or outside the project, i.e., in other projects. A more details study with statistics of code reuse in open source software is conducted. FOSS repositories such as SourceForge offer huge amounts of reusable code.

Distribution and Licensing

Internet is the medium of distribution of Free and Open Source Software. Download websites, mailing-lists, blogs, forums, etc., all contribute to the wide spread publicity and distribution of Free and Open Source Software. Wide ranges of licensing options, such as GPL, LGPL, BSD,

ISC, Artistic License, etc., are available for FOSS distribution.

Reward Mechanisms

At the beginning of Free Software movement, seemingly it was difficult to perceive the business opportunities of Free and Open Source Software. But now business model of FOSS is getting success. Sources of income range from donations to providing services such as consulting, integration, support and training. It also worth to mention that reward other than money, such as reputation and serving community is also important for many developers.

THE OSS DEVELOPMENT PROCESS

Open-source software development is the process by which open-source software, or similar software whose source code is publicly available, is developed. These are software products available with its source code under an open-source license to study, change, and improve its design. Examples of some popular open-source software products are Mozilla Firefox, Google Chromium, Android, LibreOffice and the VLC media player. Open-source software development has been a large part of the creation of the World Wide Web as we know it, with Tim Berners-Lee contributing his HTML code development as the original platform upon which the internet is now built.

In his 1997 essay *The Cathedral and the Bazaar*, open-source evangelist Eric S. Raymond suggests a model for developing OSS known as the *bazaar* model. Raymond likens the development of software by traditional methodologies to building a cathedral, "carefully crafted by individual wizards or small bands of mages working in splendid isolation". He suggests that all software should be developed using the bazaar style, which he described as "a great babbling bazaar of differing agendas and approaches."

In the traditional model of development, which he called the *cathedral* model, development takes place in a centralized way. Roles are clearly defined. Roles include people

dedicated to designing (the architects), people responsible for managing the project, and people responsible for implementation. Traditional software engineering follows the cathedral model.

The bazaar model, however, is different. In this model, roles are not clearly defined. Gregorio Robles suggests that software developed using the bazaar model should exhibit the following patterns:

Users should be treated as co-developers

The users are treated like co-developers and so they should have access to the source code of the software. Furthermore, users are encouraged to submit additions to the software, code fixes for the software, bug reports, documentation etc. Having more co-developers increases the rate at which the software evolves. Linus's law states, "Given enough eyeballs all bugs are shallow." This means that if many users view the source code, they will eventually find all bugs and suggest how to fix them. Note that some users have advanced programming skills, and furthermore, each user's machine provides an additional testing environment. This new testing environment offers that ability to find and fix a new bug.

Early releases

The first version of the software should be released as early as possible so as to increase one's chances of finding co-developers early.

Frequent integration

Code changes should be integrated (merged into a shared code base) as often as possible so as to avoid the overhead of fixing a large number of bugs at the end of the project life cycle. Some open source projects have nightly builds where integration is done automatically on a daily basis.

Several versions

There should be at least two versions of the software. There should be a buggier version with more features and a more stable version with fewer features. The buggy version (also called the development version) is for users who want the immediate use of the latest features, and are willing to accept the risk of using code that is not yet thoroughly tested. The users can then act as co-developers, reporting bugs and providing bug fixes.

High modularization

The general structure of the software should be modular allowing for parallel development on independent components.

Dynamic decision making structure

There is a need for a decision making structure, whether formal or informal, that makes strategic decisions depending on changing user requirements and other factors. Cf. Extreme programming.

Data suggests, however, that OSS is not quite as democratic as the bazaar model suggests. An analysis of five billion bytes of free/open source code by 31,999 developers shows that 74% of the code was written by the most active 10% of authors. The average number of authors involved in a project was 5.1, with the median at 2.

Open source software is usually easier to obtain than proprietary software, often resulting in increased use. Additionally, the availability of an open source implementation of a standard can increase adoption of that standard. It has also helped to build developer loyalty as developers feel empowered and have a sense of ownership of the end product.

Moreover, lower costs of marketing and logistical services are needed for OSS. OSS also helps companies keep abreast of technology developments. It is a good tool to promote a company's image, including its commercial products. The OSS development approach has helped produce reliable, high quality software quickly and inexpensively.

Open source development offers the potential for a more flexible technology and quicker innovation. It is said to be more reliable since it typically has thousands of independent programmers testing and fixing bugs of the software. Open source is not dependent on the company or author that originally created it. Even if the company fails, the code continues to exist and be developed by its users. Also, it uses open standards accessible to everyone; thus, it does not have the problem of incompatible formats that exist in proprietary software.

It is flexible because modular systems allow programmers to build custom interfaces, or add new abilities to it and it is innovative since open source programs are the product of collaboration among a large number of different programmers. The mix of divergent perspectives, corporate objectives, and personal goals speeds up innovation.

Moreover, free software can be developed in accord with purely technical requirements. It does not require thinking about commercial pressure that often degrades the quality of the software. Commercial pressures make traditional software developers pay more attention to customers' requirements than to security requirements, since such features are somewhat invisible to the customer.

It is sometimes said that the open source development process may not be well defined and the stages in the development process, such as system testing and documentation may be ignored. However this is only true for small (mostly single programmer) projects. Larger, successful projects do define and enforce at least some rules as they need them to make the teamwork possible. In the most complex projects these rules may be as strict as reviewing even minor change by two independent developers.

Not all OSS initiatives have been successful, for example SourceXchange and Eazel. Software experts and researchers who are not convinced by open source's ability to produce quality systems identify the unclear process, the late defect discovery and the lack of any empirical evidence as the most important problems (collected data concerning productivity and quality). It is also difficult to design a commercially sound business model around the open source paradigm. Consequently, only technical requirements may be satisfied and not the ones of the market. In terms of security, open source may allow hackers to know about the weaknesses or loopholes of the software more easily than closed-source software. It depends on control mechanisms in order to create effective performance of autonomous agents who participate in virtual organizations.

Development tools

In OSS development, tools are used to support the development of the product and the development process itself.

Revision control systems such as Concurrent Versions System (CVS) and later Subversion (SVN) and Git are examples of tools, often themselves open source, help manage the source code files and the changes to those files for a software project. The projects are frequently hosted and published on sites like Launchpad, Bitbucket, and GitHub.

Open source projects are often loosely organized with "little formalised process modelling or support", but utilities such as issue trackers are often used to organize open source software development. Commonly used bugtrackers includeBugzilla and Redmine.

Tools such as mailing lists and IRC provide means of coordination among developers. Centralized code hosting sites also have social features that allow developers to communicate.

TABOOS AND NORMS IN OSS DEVELOPMENT

Formalized project management, in the conventional software engineering sense, does not typically apply in OSS development. Because the development pool spans great geographic and cultural space, face-to-face "meetings" are rare. Furthermore, in noncommercial OSS development (the majority) there's no organizational bottom-line to consider; nor are there any sanctions in terms of the possibility of firing developers.

However, to avoid chaos, there are some cultural norms that govern how OSS projects are managed. Some of these are in the form of taboos. Chief among these is probably the desire to avoid projects splitting into rival and competing development streams, termed *forking* in the OSS community (Raymond, 2001HtN).

The rationale for the desire to avoid forking is clear, as contributors cannot realistically contribute to multiple forks of the same product simultaneously. Also, even with a large development pool, this is extremely wasteful. (*Note:* Forking is different to the parallel development phenomenon of competition among solutions as described above.) Since the OSS culture is driven by a reputation model, the temptation for forking must always be high. For example, you get the most attention from being the project leader, and thus contributing to a project someone else leads is always contributing more to someone else's reputation than your own. This is not altogether palatable in an ego-based economy.

Another OSS taboo relates to plagiarizing work as your own by removing the credit to the rightful contributors. This recognition of developer contribution at the micro level of individual modules is vital to ensure that developers are motivated to continue contributing - it represents rapid feedback that your contribution is valued, and this type of rapid recognition generally does not occur in a traditional software development environment.

Likewise, at a macro level, the whole OSS concept is premised on the assumption that pirate developers or organizations will not simply steal the source code that has been made available to them, and convert it to a proprietary closed source product. Thus, hijacking the work of others is a very serious taboo in OSS. Jorgensen (2001) discusses the importance of this in the FreeBSD Project where maintenance responsibility for modules is typically the responsibility of a single individual listed in the log file. The FreeBSD community have enshrined this principle as a rule, "Respect existing maintainers if listed" (FreeBSD, 2001).

Another norm that appears to be very important in the OSS community is modesty and self-deprecation on the part of developers. This is vital if contributions from others are to be solicited. If the original developer conveys the impression that no help is needed, then contributions are not likely to be very forthcoming. There are many examples of this phenomenon in OSS.

THE OSS DEVELOPMENT LIFE CYCLE

Open-source software development can be divided into several phases. The phases specified here are derived from *Sharma et al*. A diagram displaying the process-data structure of open-source software development is shown on the right. In this picture, the phases of open-source software development are displayed, along with the corresponding data elements. This diagram is made using the meta-modeling and meta-process modeling techniques.

Starting an open-source project

There are several ways in which work on an open-source project can start:

1. An individual who senses the need for a project announces the intent to develop a project in public.
2. A developer working on a limited but working codebase, releases it to the public as the first version of an open-source program.

3. The source code of a mature project is released to the public.
4. A well-established open-source project can be forked by an interested outside party.

Eric Raymond observed in his essay *The Cathedral and the Bazaar* that announcing the intent for a project is usually inferior to releasing a working project to the public.

It's a common mistake to start a project when contributing to an existing similar project would be more effective (NIH syndrome). To start a successful project it is very important to investigate what's already there. The process starts with a choice between the adopting of an existing project, or the starting of a new project. If a new project is started, the process goes to the Initiation phase. If an existing project is adopted, the process goes directly to the Execution phase.

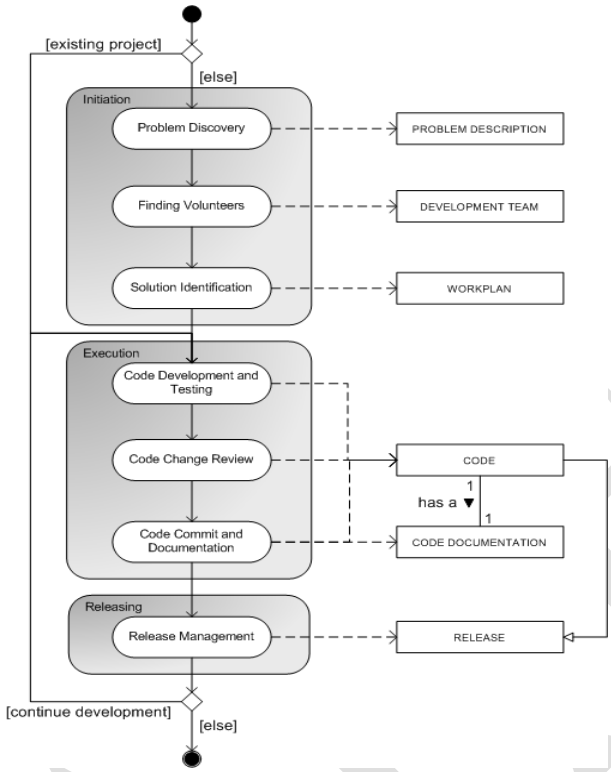
Types of open-source projects

Several types of open-source projects exist. First, there is the garden variety of software programs and libraries, which consist of standalone pieces of code. Some might even be dependent on other open-source projects. These projects serve a specified purpose and fill a definite need. Examples of this type of project include the Linux kernel, the Firefox web browser and the LibreOffice office suite of tools.

Distributions are another type of open-source project. Distributions are collections of software that are published from the same source with a common purpose. The most prominent example of a "distribution" is an operating system. There are many Linux distributions (such as Debian, Fedora Core, Mandriva, Slackware, Ubuntu etc.) which ship the Linux kernel along with many user-land components. There are other distributions, like ActivePerl, the Perl programming language for various operating systems, and Cygwin distributions of open-source programs for Microsoft Windows.

Other open-source projects, like the BSD derivatives, maintain the source code of an entire operating system, the kernel and all of its core components, in one revision controlsystem; developing the entire system together as a single team. These operating system development projects closely integrate their tools, more so than in the other distribution-based systems.

Finally, there is the book or standalone document project. These items usually do not ship as part of an open-source software package. The Linux Documentation Project hosts many such projects that document various aspects of the GNU/Linux operating system. There are many other examples of this type of open-source project.



Methods

It is hard to run an open-source project following a more traditional software development method like the waterfall model, because in these traditional methods it is not allowed to go back to a previous phase. In open-source software development, requirements are rarely gathered before the start of the project; instead they are based on early releases of the software product, as Robbins describes. Besides requirements, often volunteer staff is attracted to help develop the software product based on the early releases of the software. The community is very harsh, much like the business world of closed-source software: "if you find the customers you survive, but without customers you die".

More generally, all Agile programming methods are applicable to open-source software development, because of their iterative and incremental character. Other Agile method are equally useful for both open and closed source software development :Internet-Speed Development, for example is suitable for open-source software development because of the distributed development principle it adopts. Internet-Speed Development uses geographically distributed teams to 'work around the clock'. This method, mostly adopted by large closed-source firms, (because they're the only ones which afford development centers in different time zones), works equally well in open source projects because a software developed by a large group of volunteers shall naturally tend to have developers spread across all time zones.

Communication channels

Developers and users of an open-source project are not all necessarily working on the project in proximity. They require some electronic means of communications. E-mail is one of the most common forms of communication among open-source developers and users. Often, electronic mailing lists are used to make sure e-mail messages are delivered to all interested parties at once.

This ensures that at least one of the members can reply to it. In order to communicate in real time, many projects use an instant messaging method such as IRC. Web forums have recently become a common way for users to get help with problems they encounter when using an open-source product. Wikis have become common as a communication medium for developers and users.

Version control systems

In OSS development the participants, who are mostly volunteers, are distributed amongst different geographic regions so there is need for tools to aid participants to collaborate in the development of source code.

During early 2000s, Concurrent Versions System (CVS) was a prominent example of a source code collaboration tool being used in OSS projects. CVS helps manage the files and codes of a project when several people are working on the project at the same time. CVS allows several people to work on the same file at the same time. This is done by moving the file into the users' directories and then merging the files when the users are done. CVS also enables one to easily retrieve a previous version of a file. During mid 2000s, The Subversion revision control system (SVN) was created to replace CVS. It is quickly gaining ground as an OSS project version control system.

Many open-source projects are now using distributed revision control systems, which scale better than centralized repositories such as SVN and CVS. Popular examples are git, used by the Linux kernel, and Mercurial, used by the Python programming language.

Bug trackers and task lists

Most large-scale projects require a bug tracking system to keep track of the status of various issues in the development of the project. Some bug trackers include:

- Bugzilla – a sophisticated web-based bug tracker from Mozilla.
- Mantis Bug Tracker – a web-based PHP/MySQL bug tracker.
- Trac – integrating a bug tracker with a wiki, and an interface to the Subversion version control system.
- Redmine – written in Ruby, integrates issue tracking, wiki, forum, news, roadmap, gantt project planning and interfaces with LDAP user directory.
- Request tracker – written in Perl. Given as a default to CPAN modules – see rt.cpan.org.
- SourceForge and its forks provide a bug tracker as part of its services. As a result, many projects hosted at SourceForge.net and similar services default to using it.
- JIRA – Atlassian's project management and issue tracking tool.

Testing and debugging tools

Since OSS projects undergo frequent integration, tools that help automate testing during system integration are used. An example of such tool is Tinderbox. Tinderbox enables participants in an OSS project to detect errors during system integration. Tinderbox runs a continuous build process and informs users about the parts of source code that have issues and on which platform(s) these issues arise.

A debugger is a computer program that is used to debug (and sometimes test or optimize) other programs. GNU Debugger (GDB) is an example of a debugger used in open-source software development. This debugger offers remote debugging, what makes it especially applicable to open-source software development.

A memory leak tool or memory debugger is a programming tool for finding memory leaks and buffer overflows. A memory leak is a particular kind of unnecessary memory consumption by a computer program, where the program fails to release memory that is no longer needed. Examples of memory leak detection tools used by Mozilla are the XPCOM Memory Leak tools. Validation tools are used to check if pieces of code conform to the specified syntax. An example of a validation tool is Splint.

DERIVING A FRAMEWORK FOR ANALYZING OSS

Zachman has drawn on the disciplines of architecture and engineering to derive a framework for IS architecture which basically contains the categories what, how, where, who, when, and why (a set of descriptors which appear to have been borrowed from Kipling). Zachman discusses in some detail the descriptors what, how, and where to categorize different IS architectures and suggests that these are independent but "inextricably linked," and suggests that, for the sake of logical completeness, these should be complemented by who, when and why (Zachman 1987; Sowa and Zachman 1992). Interestingly, Zachman concluded the 1987 paper by suggesting that the framework could be used in a number of areas, including to rethink the nature of software development.

A Framework for Analyzing the OSS Approach

What (Transformation)

• What defines a software project as OSS?

• What types of projects tend to be OSS?

Why (Weltanshauung, or World View)

• What are the technological motivations for OSS development?

• What are the economic motivations for OSS development?

• What are the socio-political motivations for OSS development?

When and Where (Environment)

• What are the temporal dimensions of OSS development?

• What are the spatial/geographic dimensions of OSS development?

How

• How is the OSS development process organized?

• What tools are used to support the OSS model?

Who (Client, Actor, Owner)

• What are the characteristics of the individual developers contributing to OSS projects?

• What are the characteristics of the companies distributing OSS products?

• What are the characteristics of the users of OSS products?

ANALYSIS OF THE OSS APPROACH

In this section, the OSS approach is analyzed in detail using each of the categories of the derived framework.

What (Transformation)

What defines a software product/project as OSS?

Open Source Software is strictly defined by the license under which it is distributed

(i.e., compliance with the OSI Open Source Definition). However, as with any emerging concept, there is some fluidity. Thus, OSS is further characterized by the dynamics described subsequently in this framework analysis.

What types of products/projects tend to be OSS?

OSS has in the past been dominated by operating and networking systems software, utilities, development tools, and infrastructural components. Currently, an increasing number of productivity and entertainment applications are being developed.

Why (Weltanschauung, or World View)

What are the technological motivations for OSS development?

The primary technological drivers for OSS include the need for more robust code, faster development cycles, higher standards of quality, reliability and stability, and more open standards/platforms.

What are the economic motivations for OSS development?

Business drivers for OSS include the corporate need for shared cost and shared risk, and the redefinition of software industry as a commodity and service industry.

What are the socio-political motivations for OSS development?

"Human" motivations for OSS include scratching a developer's "personal itch," the desire for advancement through mentorship, peer reputation, the desire for "meaningful" work, and community oriented idealism.

When and Where (Environment)

What are the temporal dimensions of OSS development?

OSS is characterised by the rapid development and rapid evolution of software, by frequent, incremental release, and by interaction in "Internet time."

What are the spatial/geographic dimensions of OSS development?

OSS is characterised by distributed developer teams, bounded by "cyberspace" rather than physical geography.

How is the OSS development process organized?

The core methodology of OSS is massive parallel development and debugging. This has traditionally involved loosely-centralized, cooperative, and gratis contribution from individual developers (although there is a recent increase in paid, coordinated development).

What tools are used to support the OSS model? Massive parallel development methods are supported by the Internet as a communication, collaboration, and distribution platform, and by concurrent versioning software. Other support tools include academic, non-profit, and commercial patrons of OSS projects, "reverse auctions" and online agency services.

Who (Client, Actor, Owner)

What are the characteristics of the individual developers contributing to OSS projects?

OSS developers have traditionally been self-identified hackers (not "crackers"), professional developers (not amateurs), self-selected, and highly-motivated. Also, paradoxically, given the reputation-based culture, developers tend to be (publicly) modest and self-deprecating. This has important implications for stimulating cooperative development.

What are the characteristics of the companies distributing OSS products?

OSS companies have had a fantastic IPO stock market history. Generally, these companies provide patronage for OSS developers akin to a Renaissance model. Profitability hinges on a paradigm shift in software development. Customer service, brand management, and peer reputation are critical. Parallels exist with the automobile and legal industries and with academe.

What are the characteristics of the users of OSS products?

To date, OSS users have primarily been expert users and early adopters. Traditionally there has been considerable overlap between the developer and user pool. As more

OSS projects focus on usability and interface issues, this profile will change.

Examples of OSS Products

Product Description

• Linux • Operating System

• Apache • Web server

• Sendmail • Internet mail utility

• BIND • Berkeley Internet Name Daemon, the software that runs the Domain

Name Server (DNS) for the Web

• PERL, Python • Programming languages

• GNU Software • A variety of compilers, utilities, and notably, the EMACS Editor

• GNOME, KDE • GUI interfaces for Linux

• Mozilla • OSS version of Netscape Navigator

• Jikes • Java compiler from IBM

• GIMP • Image workshop

• Darwin • Mac OSX server system

• SAMBA • Allows integration with Microsoft's SMB protocol

• Ghostcript • Aladdin enterprises PDF utility

• Doom • First-person shooter game

OSS products are typically development tools, back-office services/applications, and infrastructural and networking utilities. Performance and reliability are critical factors and these products score very highly on these criteria. They have been chosen by technically-aware IT personnel who are not as susceptible to a glossy marketing campaign as the wider market.

This reflects the OSS-like nature of the evolution of the Internet, e.g., the TCP/IP protocol, DNS, and electronic mail utilities. Also, the initial implementation of the Web was characterized by an OSS style; thus, it is appropriate that utilities such as BIND and the Apache web server should be OSS. It should be noted that as OSS-related companies seek to push the movement into the commercial mainstream, entertainment and end-user productivity tools are becoming more popular.

ZACHMAN'S FRAMEWORK FOR IS ARCHITECTURE

The title "Zachman Framework" refers to The Zachman Framework for Enterprise Architecture with version 3.0 being the most current. The Zachman Framework has evolved in its thirty-year history to include:

- The initial framework, named *A Framework for Information Systems Architecture*, by John Zachman published in an 1987 article in the IBM Systems journal.
- The *Zachman Framework for Enterprise Architecture*, an update of the 1987 original in the 1990s extended and renamed .
- One of the later versions of the Zachman Framework, offered by Zachman International as industry standard.

Collage of Zachman Frameworks as presented in several books on Enterprise Architecture from 1997 to 2005.

In other sources the Zachman Framework is introduced as a framework, originated by and named after John Zachman, represented in numerous ways, see image. This framework is explained as, for example:

- a framework to organize and analyze data
- a framework for enterprise architecture.
- a classification system, or classification scheme
- a matrix, often in a 6x6 matrix format
- a two-dimensional model[10] or an analytic model.
- a two-dimensional schema, used to organize the detailed representations of the enterprise.

Beside the frameworks developed by John Zachman, numerous extensions and/or applications have been developed, which are also sometimes called Zachman Frameworks, however they generally tend to be graphical overlays of the actual framework itself.

The Zachman Framework summarizes a collection of perspectives involved in enterprise architecture. These perspectives are represented in a two-dimensional matrix that defines along the rows the type of stakeholders and with the columns the aspects of the architecture. The framework does not define a methodology for an architecture. Rather, the matrix is a template that must be filled in by the goals/rules, processes, material, roles, locations, and events specifically required by the organization. Further modeling by mapping between columns in the framework identifies gaps in the documented state of the organization.

The framework is a logical structure for classifying and organizing the descriptive representations of an enterprise. It is significant to both the management of the enterprise, and the actors involved in the development of enterprise systems.[13]While there is no order of priority for the columns of the Framework, the top-down order of the rows is significant to the alignment of business concepts and the actual physical enterprise. The level of detail in the Framework is a function of each cell (and not the rows). When done by IT the lower level of focus is on information technology, however it can apply equally to physical material (ball valves, piping, transformers, fuse boxes for example) and the associated physical processes, roles, locations etc. related to those items.

Concept

The basic idea behind the Zachman Framework is that the same complex thing or item can be described for different purposes in different ways using different types of descriptions (e.g., textual, graphical). The Zachman Framework provides the thirty-six necessary categories for completely describing anything; especially complex things like manufactured goods (e.g., appliances), constructed structures (e.g., buildings), and enterprises (e.g., the organization and all of its goals, people, and technologies). The framework provides six different transformations of an abstract idea (not increasing in detail, but transforming) from six different perspectives.

It allows different people to look at the same thing from different perspectives. This creates a holistic view of the environment, an important capability illustrated in the figure.

Views of rows

Each row represents a total view of the solution from a particular perspective. An upper row or perspective does not necessarily have a more comprehensive understanding of the whole than a lower perspective. Each row represents a distinct, unique perspective; however, the deliverables from each perspective must provide sufficient detail to define the solution at the level of perspective and must translate to the next lower row explicitly.

Each perspective must take into account the requirements of the other perspectives and the restraint those perspectives impose. The constraints of each perspective are additive. For example, the constraints of higher rows affect the rows below. The constraints of lower rows can, but do not necessarily affect the higher rows. Understanding the requirements and constraints necessitates communication of knowledge and understanding from perspective to perspective. The Framework points the vertical direction for that communication between perspectives.

- *Executive Perspective* (Scope Contents) - The first architectural sketch is a "bubble chart" orVenn diagram, which depicts in gross terms the size, shape, partial relationships, and basic purpose of the final structure. It corresponds to an executive summary for a planner or investor who wants an overview or estimate of the scope of the system, what it would cost, and how it would relate to the general environment in which it will operate.
- *Business Management Perspective* (Business Concepts) - Next are the architect's drawings that depict the final building from the perspective of the owner, who will have to live with it in the daily routines of business. They correspond to the enterprise (business) models, which constitute the designs of the business and show the business entities and processes and how they relate.
- *Architect Perspective* (System Logic) - The architect's plans are the translation of the drawings into detail requirements representations from the designer's perspective. They correspond to the system model designed by a systems analyst who must determine the data elements, logical process flows, and functions that represent business entities and processes.
- *Engineer Perspective* (Technology Physics) - The contractor must redraw the architect's plans to represent the builder's perspective, with sufficient detail to understand the constraints of tools, technology, and materials. The builder's plans correspond to the technology models, which must adapt the information systems model to the details of the programming languages, input/output (I/O) devices, or other required supporting technology.

- *Technician Perspective* (Tool Components) - Subcontractors work from shop plans that specify the details of parts or subsections. These correspond to the detailed specifications that are given to programmers who code individual modules without being concerned with the overall context or structure of the system. Alternatively, they could represent the detailed requirements for various commercial-off-the-shelf (COTS), government off-the-shelf (GOTS), or components of modular systems software being procured and implemented rather than built.
- *Enterprise Perspective* or (Operations Instances)

Focus of columns

In summary, each perspective focuses attention on the same fundamental questions, then answers those questions from that viewpoint, creating different descriptive representations (i.e., models), which translate from higher to lower perspectives. The basic model for the focus (or product abstraction) remains constant. The basic model of each column is uniquely defined, yet related across and down the matrix. In addition, the six categories of enterprise architecture components, and the underlying interrogatives that they answer, form the columns of the Zachman Framework and these are:

1. Inventory Sets — What
2. Process Flows — How
3. Distribution Networks — Where
4. Responsibility Assignments — Who
5. Timing Cycles — When
6. Motivation Intentions — Why

In Zachman's opinion, the single factor that makes his framework unique is that each element on either axis of the matrix is explicitly distinguishable from all the other elements on that axis. The representations in each cell of the matrix are not merely successive levels of increasing detail, but actually are different representations — different in context, meaning, motivation, and use. Because each of the elements on either axis is explicitly different from the others, it is possible to define precisely what belongs in each cell.

Models of cells

The Zachman Framework typically is depicted as a bounded 6 x 6 "matrix" with the Communication Interrogatives as Columns and the Reification Transformations as Rows. The framework classifications are repressed by the Cells, that is, the intersection between the Interrogatives and the Transformations.

The cell descriptions are taken directly from version 3.0 of the Zachman Framework.

Executive Perspective

1. (What) Inventory Identification
2. (How) Process Identification
3. (Where) Distribution Identification
4. (Who) Responsibility Identification
5. (When) Timing Identification
6. (Why) Motivation Identification

Business Management Perspective

1. (What) Inventory Definition
2. (How) Process Definition
3. (Where) Distribution Definition
4. (Who) Responsibility Definition
5. (When) Timing Definition
6. (Why) Motivation Definition

Architect Perspective

1. (What) Inventory Representation
2. (How) Process Representation
3. (Where) Distribution Representation
4. (Who) Responsibility Representation
5. (When) Timing Representation
6. (Why) Motivation Representation

Engineer Perspective

1. (What) Inventory Specification
2. (How) Process Specification
3. (Where) Distribution Specification
4. (Who) Responsibility Specification
5. (When) Timing Specification
6. (Why) Motivation Specification

Technician Perspective

1. (What) Inventory Configuration
2. (How) Process Configuration
3. (Where) Distribution Configuration
4. (Who) Responsibility Configuration
5. (When) Timing Configuration
6. (Why) Motivation Configuration

Enterprise Perspective

1. (What) Inventory Instantiations
2. (How) Process Instantiations
3. (Where) Distribution Instantiations
4. (Who) Responsibility Instantiations
5. (When) Timing Instantiations
6. (Why) Motivation Instantiations

Since the product development (i.e., architectural artifact) in each cell or the problem solution embodied by the cell is the answer to a question from a perspective, typically, the models or descriptions are higher-level depictions or the surface answers of the cell. The refined models or designs supporting that answer are the detailed descriptions within the cell. Decomposition (i.e., drill down to greater levels of detail) takes place within each cell. If a cell is not made explicit (defined), it is implicit (undefined). If it is implicit, the risk of making assumptions about these cells exists. If the assumptions are valid, then time and money are saved. If, however, the assumptions are invalid, it is likely to increase costs and exceed the schedule for implementation.

Framework set of rules

Example of Zachman Framework Rules.

The framework comes with a set of rules:

- *Rule 1 The columns have no order* : The columns are interchangeable but cannot be reduced or created
- *Rule 2 Each column has a simple generic model* : Every column can have its own meta-model
- *Rule 3 The basic model of each column must be unique* : The basic model of each column, the relationship objects and the structure of it is unique. Each relationship object is interdependent but the representation objective is unique.

- *Rule 4 Each row describes a distinct, unique perspective* : Each row describes the view of a particular business group and is unique to it. All rows are usually present in most hierarchical organizations.
- *Rule 5 Each cell is unique* : The combination of 2,3 & 4 must produce unique cells where each cell represents a particular case. Example: A2 represents business outputs as they represent what are to be eventually constructed.
- *Rule 6 The composite or integration of all cell models in one row constitutes a complete model from the perspective of that row* : For the same reason as for not adding rows and columns, changing the names may change the fundamental logical structure of the Framework.
- *Rule 7 The logic is recursive* : The logic is relational between two instances of the same entity.

The framework is generic in that it can be used to classify the descriptive representations of any physical object as well asconceptual objects such as enterprises. It is also recursive in that it can be used to analyze the architectural composition of itself. Although the framework will carry the relation from one column to the other, it is still a fundamentally structural representation of the enterprise and not a flow representation.

Flexibility in level of detail

One of the strengths of the Zachman Framework is that it explicitly shows a comprehensive set of views that can be addressed by enterprise architecture.[12] Some feel that following this model completely can lead to too much emphasis on documentation, as art efacts would be needed for every one of the thirty cells in the framework. Zachman, however, indicates that only the facts needed to solve the problem under analysis need be populated.

John Zachman clearly states in his documentation, presentations, and seminars that, as framework, there is flexibility in what depth and breadth of detail is required for each cell of the matrix based upon the importance to a given organization. An automaker whose business goals may necessitate an inventory and process-driven focus, could find it beneficial to focus their documentation efforts on What and How columns. By contrast, a travel agent company, whose business is more concerned with people and event-timing, could find it beneficial to focus their documentation efforts on Who, When, andWhere columns. However, there is no escaping the Why column's importance as it provides the business drivers for all the other columns.

Applications and influences

Since the 1990s the Zachman Framework has been widely used as a means of providing structure for Information Engineering-style enterprise modeling. The Zachman Framework can

be applied both in commercial companies and in government agencies. Within a government organization the framework can be applied to an entire agency at an abstract level, or it can be applied to various departments, offices, programs, subunits and even to basic operational entities.

Standards based on the Zachman Framework

Zachman Framework is also used as a framework to describe standards, for example standards for healthcare and healthcare information system. Each cell of the framework contains such a series of standards for healthcare and healthcare information system.

While the Zachman Framework is widely discussed, its practical value has been questioned:

- The framework is purely speculative, non-empirical and based only on the conceptual argument that the "equivalency [between the architectural representations of the manufacturing and construction industries] would strengthen the argument that an analogous set of architectural representations is *likely* to be produced during the process of building any complex engineering product, including an information system"
- Practical feedback shows that the general idea of creating comprehensive descriptions of enterprises as suggested by the Zachman Framework is unrealistic
- In 2004 John Zachman admitted that the framework is theoretical and has never been fully implemented: "If you ask who is successfully implementing the whole framework, the answer is nobody that we know of yet"
- There are no detailed examples demonstrating the successful practical application of the framework
- EA practitioner Stanley Gaver argues that "the analogy to classical architecture first made by John Zachman is faulty and incomplete"
- Jason Bloomberg argues that "enterprise isn't an ordinary system like a machine or a building, and can't be architected or engineered as such"

This criticism suggests that the Zachman Framework can hardly reflect actual best practice in EA.

CATWOE AND SOFT SYSTEM METHOD

The methodology was developed from earlier systems engineering approaches, primarily by Peter Checkland and colleagues such as Brian Wilson. The primary use of SSM is in the analysis of complex situations where there are divergent views about the definition of the problem. These situations are "soft problems" such as: How to improve health services delivery? How to manage disaster planning? When should mentally disordered offenders be diverted from custody? What to do about homelessness amongst young people?

In such situations even the actual problem to be addressed may not be easy to agree upon. To intervene in such situations the soft systems approach uses the notion of a "system" as an interrogative device that will enable debate amongst concerned parties. In its 'classic' form the methodology consists of seven steps, with initial appreciation of the problem situation leading to the modelling of several human activity systems that might be thought relevant to the problem situation.

There are several hundred documented examples of the successful use of SSM in many different fields, ranging from ecology, to business and military logistics. It has been adopted by many organizations and incorporated into other approaches: in the 1990s for example it was the recommended planning tool for the UK government's SSADM system development methodology.

The general applicability of the approach has led to some criticisms that it is functionalist, non-emancipator or supports the status quo and existing power structures; this is a claim that users would deny, arguing that the methodology itself can be none of these, it is the user of the methodology that may choose to employ it in such a way.

The methodology has been described in several books and many academic articles.

SSM remains the most widely used and practical application of systems thinking, and other systems approaches such as critical systems thinking have incorporated many of its ideas.

The 7-stage description

7-stage representation of SSM:

1. Enter situation considered problematical
2. Express the problem situation
3. Formulate root definitions of relevant systems of purposeful activity
4. Build conceptual models of the systems named in the root definitions
5. Compare models with real world situations
6. Define possible changes which are both possible and feasible
7. Take action to improve the problem situation

CATWOE

In 1975, David Smyth, a researcher in Checkland's department, observed that SSM was most successful when the Root Definition included certain elements. These elements, captured in the mnemonic CATWOE, identified the people, processes and environment that contribute to a situation, issue or problem that required analyzing.

This is used to prompt thinking about what the business is trying to achieve. Business Perspectives help the Business Analyst to consider the impact of any proposed solution on the people involved. There are six elements of CATWOE

Customers - Who are the beneficiaries of the highest level business process and how does the issue affect them?

Actors - Who is involved in the situation, who will be involved in implementing solutions and what will impact their success?

Transformation Process - What is the transformation that lies at the heart of the system - transforming grapes into wine, transforming unsold goods into sold goods, transforming a societal need into a societal need met?

Weltanschuung (or Worldview) - What is the big picture and what are the wider impacts of the issue?

Owner - Who owns the process or situation being investigated and what role will they play in the solution?

Environmental Constraints - What are the constraints and limitations that will impact the solution and its success?

DERIVING THE ANALYTICAL FRAMEWORK FOR OSS

A Framework for Analyzing the OSS Approach

What (Transformation)

• What defines a software project as OSS?

• What types of projects tend to be OSS?

Why (Weltanshauung, or World View)

• What are the technological motivations for OSS development?

• What are the economic motivations for OSS development?

• What are the socio-political motivations for OSS development?

When and Where (Environment)

• What are the temporal dimensions of OSS development?

• What are the spatial/geographic dimensions of OSS development?

How

• How is the OSS development process organized?

• What tools are used to support the OSS model?

Who (Client, Actor, Owner)

• What are the characteristics of the individual developers contributing to OSS projects?

• What are the characteristics of the companies distributing OSS products?

• What are the characteristics of the users of OSS products?

ANALYSIS OF THE OSS APPROACH

In this section, the OSS approach is analyzed in detail using each of the categories of the derived framework.

What (Transformation)

What defines a software product/project as OSS?

Open Source Software is strictly defined by the license under which it is distributed

(i.e., compliance with the OSI Open Source Definition). However, as with any emerging concept, there is some fluidity. Thus, OSS is further characterized by the dynamics described subsequently in this framework analysis.

What types of products/projects tend to be OSS?

OSS has in the past been dominated by operating and networking systems software, utilities, development tools, and infrastructural components. Currently, an increasing number of productivity and entertainment applications are being developed.

Why (Weltanschauung, or World View)

What are the technological motivations for OSS development?

The primary technological drivers for OSS include the need for more robust code, faster development cycles, higher standards of quality, reliability and stability, and more open standards/platforms.

What are the economic motivations for OSS development?

Business drivers for OSS include the corporate need for shared cost and shared risk, and the redefinition of software industry as a commodity and service industry.

What are the socio-political motivations for OSS development?

"Human" motivations for OSS include scratching a developer's "personal itch," the desire for advancement through mentorship, peer reputation, the desire for "meaningful" work, and community oriented idealism.

When and Where (Environment)

What are the temporal dimensions of OSS development?

OSS is characterised by the rapid development and rapid evolution of software, by frequent, incremental release, and by interaction in "Internet time."

What are the spatial/geographic dimensions of OSS development?

OSS is characterised by distributed developer teams, bounded by "cyberspace" rather than physical geography.

How is the OSS development process organized?

The core methodology of OSS is massive parallel development and debugging. This has traditionally involved loosely-centralized, cooperative, and gratis contribution from individual developers (although there is a recent increase in paid, coordinated development).

What tools are used to support the OSS model? Massive parallel development methods are supported by the Internet as a communication, collaboration, and distribution platform, and by

concurrent versioning software. Other support tools include academic, non-profit, and commercial patrons of OSS projects, "reverse auctions" and online agency services.

Who (Client, Actor, Owner)

What are the characteristics of the individual developers contributing to OSS projects?

OSS developers have traditionally been self-identified hackers (not "crackers"), professional developers (not amateurs), self-selected, and highly-motivated. Also, paradoxically, given the reputation-based culture, developers tend to be (publicly) modest and self-deprecating. This has important implications for stimulating cooperative development.

POSSIBLE QUESTIONS

(8 MARKS)

1. Briefly describe about OSS development life cycle.
2. Explain the characteristics of Open Source Software Transformation
3. Describe about Deriving a framework for analyzing OSS.
4. Describe Zachman‟s Framework for IS Architecture.
5. Briefly describe about qualification and categorizing open source software in detail.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
COIMBATORE-641021
Department of Computer Science
III B.Sc(CS) (BATCH 2015-2018)
Open Source Software
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS
ONLINE EXAMINATIONS **ONE MARK QUESTIONS**

UNIT II

S.No	Question	Option1	Option2	Option3	Option4			Answer
1	_____ is a software that is floated to see if there is an interest	Vaporware	Freeware	Free Software	Open Source Software			Vaporware
2	The software in wide distribution that has a version number less than _____	3.0	2.0	1.0	0.5			1.0
3	OSH stands for _____	Open Source Host	Open Source Hardware	Open Source Hub	Open Source Hierarchy			Open Source Hardware
4	The development of open source hardware was initiated in _____	2004	2000	2002	2001			2002
5	F in FOSS stands for _____	feasible	full	free	field			free
6	FOSS projects break their version number into_____ pieces	4	3	2	1			3
7	The second piece of a version number of FOSS project is_____	major version	minor update	point release	minor version			minor update
8	Minor update is the _____ piece of version number of FOSS project.	first	second	third	fourth			second
9	_____ indicates a small update to fix a bug or security problem	point release	minor update	major version	minor version			point release
10	HDL stands for _____	High Definition Language	Hardware Definition Language	High Description Language	Hardware Description Language			Hardware Description Language

11	Pajamas media briefly known as_____	free software media	open hardware media	open source media	close source media			open source media
12	Open Source Media is a start up company founded in the year_____	2001	2002	2003	2004			2004
13	OST stands for_____	Open Source Teaching	Open software Teaching	Open Service Teaching	Open Server Teaching			Open Source Teaching
14	_____ is collection of just about any type of files	Object	Learning Object	Class Object	Source Object			Learning Object
15	Each file in a learning object is referred as ____	document	group	atom	record			atom
16	Each learning object must have_____ as a starting or entry point	prototype atom	sub atom	model atom	main atom			main atom
17	In_____ model, source code must be hidden from the public competitors.	open source	close source	free source	shared source			close source
18	Free software movement demands _____ types of freedoms	4	5	6	7			4
19	_____ is microsoft's framework for sharing computer source code with individuals and organizations	open source	shared source	close source	free source			shared source
20	Richard's Stallman organization is called _____	Free Software Foundation	Open Source Software Foundation	Close Source Software Foundation	Apache Software Foundation			Free Software Foundation
21	Most of the open sourc software licensed under_____	copyright	copyleft	patent	legal license			copyleft
22	_____ denotes the theory of right action and the greater good.	Ethics	Moral	Immoral	legal license			Ethics
23	_____ ethics signifies a moral code applicable to individuals	social	personal	financial	logical			personal
24	_____ ethics can be synonymous with social, political philosophy in as much as it is the foundation of a good society or state	Personal	Financial	Logical	Social			Social

25	The Wix toolset is licensed under the _____	CPL	GPL	LGPL	MPL			CPL
26	The ASP.Net AJAX control toolkit licensed under_____	MPL	MS-PL	CPL	NPL			MS-PL
27	_____ toolkit is a set of controls and extenders use AJAX technologies to enable the developers to improve the client experience on their website	ASP.Net AJAX	ASP.Net	Wix Toolset	AJAX			ASP.Net AJAX
28	Open politics is also known as_____	open software politics	open source politics	open source software politics	free software politics			open source politics
29	In Linux kernel version 2, the design use _____ numbers for Development kernels	odd	even	binary	decimal			odd
30	In Linux kernel version 2, the design use _____ numbers for stable kernels	Random	odd	even	binary			even
31	_____ is the application of open source methods to the creation of products, machines and system	Open design	Software design	Hardware design	System Design			Open design
32	EDA stands for_____	Electronic Design Automation	Electrical Display Architecture	Electronic Device Automation	Electrical Device Architecture			Electroni
33	Programmers often prefer to _____ from other's code.	Copy	Reuse	steal	learn			learn
34	What is CVS?	Open source license	Editing software	Version control system	Development Environment			Version control system
35	Which of the following is not true for open source software?	It is owned by a person	It supports distributed development	It supports collaborative development	Its code is available for all			It is owned by a person

36	The acronym of CVS is _____.	Consistent Version System	Common Validation System	Consistent Version System	Concurrent Versions System			Concurrent Versions System
37	_____ has advanced data store than CVS.	RCS	SCCS	SVN	IMS			CVS
38	The acronym of SVN is _____.	Single Version	Subversion	Stimuli Verification	Simple Verified Node			Subversion
39	Subversion is a project of _____.	IBM	Oracle	Apple	Apache			Apache
40	SourceSafe is a _____.	version control	backup	website	search engine			version control
41	What is Bugzilla?	A bug-tracking mechanism	A version control	A bug-correcting software	It involves both version control and bug-tracking mechanism			A bug-tracking mechanism
42	Which of the following is proprietary?	OpenOffice	Oracle	MySQL	Postgres			Oracle
43	Bugs persist longer in _____.	open codebases	proprietary codebases	free codebases	closed codebases			closed codebases

c Design Automation

UNIT III

SYLLABUS

ENVIRONMENT: THE "WHERE" OF OSS AND THE "WHEN" OF OSS

The developmental environment of OSS is one that is shaped by "Internet time" and virtual geography. As noted above, OSS projects are often characterized by rapid development. Perhaps more importantly, the platform-building ethos of OSS allows for rapid evolution of software, addressing what Young and Rohm (2000) identify as a sharp disparity between hardware and software innovation, in that hardware advances run at a rate of more than twice that of software.

OSS, as its history to date has shown, offers some promise that this can be redressed. Again, the huge expansion in the pool of potential co-developers has parallels with what took place in the automobile and telephone industries in the past. Fogel (1999) also discusses the parallels between OSS and biological evolution and identifies features such as natural selection and survival of the fittest from actual examples of OSS projects. The modus operandi of frequent, incremental releases encourages adaptation and mutation, and the asynchronous collaboration of developers means that OSS projects achieve an agility of which corporations often only dream. Geographically OSS is characterized by massive distribution, with teams, community, and peer groups defined by virtual, rather than physical, boundaries.

When and Where (Environment)

• What are the temporal dimensions of OSS development?

OSS is characterized by the rapid development and rapid evolution of software, by frequent, incremental release, and by interaction in "Internet time."

• What are the spatial/geographic dimensions of OSS development?

OSS is characterized by distributed developer teams, bounded by "cyberspace" rather than physical geography.

Proprietary software development projects are often characterized by the development of software products through sequential and incremental releases. A product is designed, developed,

tested, and then released to the public with upgrades usually available for download or subsequently released as new versions of the application. The software product is developed through the interaction and cooperation between software developers who can be physically close or geographically dispersed. While these projects release software sequentially and incrementally, they differ from open source projects as these are characterized by the rapid development and evolution of software marked by a fluid design state (Raymond 2001) through which continuous improvements are made to the end product and then distributed to end users. In essence, the fluid design state characterizes the ever-changing aspect of open source products as these applications are constantly being modified, improved, and released. In addition, OSS development projects are characterized by distributed software development efforts. Development teams in open source contexts are held together by the Web rather than physical geography. The Internet, as a free public communication infrastructure, has made the movement possible.

WORLD VIEW: A FRAMEWORK FOR CLASSIFYING OSS MOTIVATIONS

There are three "world views" important to understanding why, on a macro level, industry might choose to develop software according to the OSS paradigm, and, on a micro level, why individual developers would choose to participate. These are discussed in turn.

TECHNOLOGICAL MICRO-LEVEL (INDIVIDUAL) MOTIVATION

The primary technological drivers for OSS include the need for more robust code, faster development cycles, higher standards of quality, reliability and stability, and more open standards/platforms.

The technological motivation for OSS development directly relates to the software crisis, which clearly illustrates that traditional modes of development do not work very well, specifically in the areas of speed, quality, and cost of development. The OSS approach counters these tripartite aspects. The Linux operating system and other OSS products mentioned above are characterized by a very rapid development time-scale. For example, new releases of Linux were produced more than once per day in the early days of its development (Raymond 1998a). Part of the conventional wisdom of software development is captured in Brooks' fundamental law, viz., "adding manpower to a late software project makes it later" (Brooks 1975). Based on this, it was thought that complex software, which all software pretty much is (Brooks 1987), required a

disciplined and orderly approach, and that Feller and Fitzgerald 64 the communication overhead of adding extra developers negated their potential contribution to development productivity. However, the OSS community has recast this law as "given enough eye-balls, every bug looks shallow" (Raymond 1998a) to reflect the manner in which the literally global community of OSS co-developers, operating in a decentralized cooperative manner according to the principle of prompt feedback, are able to solve the various problems that arise. Linux has had more than 1,000 developers working on the kernel alone, while Fetchmail has had more than 600 globally-distributed co-developers working asynchronously in different time-zones (Raymond 1998a).

The second aspect of the software crisis, software quality, is also addressed by the OSS approach, in that OSS developers are reckoned to be the most-talented and highly motivated 5% of software developers (Raymond 1998a). Also, peer review of any development product is truly independent, in that the global community of co-developers have no vested interest, consciously or subconsciously, in turning a blind eye to deficiencies in the product. Evidence of this quality and inherent reliability of OSS output is amply demonstrated by the fact that these products have achieved such a significant market share without any conventional marketing or advertising campaigns—there has not been any $100 million Start me up campaign for any OSS products!

Raymond (1999a) has considered the specific issue of when OSS development is appropriate in some detail and suggests the following (mainly technological criteria) as predisposing toward OSS development:

• When reliability and stability of the software are critical,

• When correctness is only established through independent peer review,

• When software is critical to the business,

• When the software establishes a communications infrastructure,

• When the key algorithms are part of common software engineering knowledge

ECONOMIC MICRO-LEVEL AND MACRO-LEVEL (INDIVIDUAL) MOTIVATION

Business drivers for OSS include the corporate need for shared cost and shared risk, and the redefinition of software industry as a commodity and service industry.

The final aspect of the software crisis, cost, is also addressed by OSS, and sets the stage for an understanding of the economic motivations of OSS development. Under the OSS model, software may be downloaded freely by ftp, or more typically, can be purchased in CD-ROM format for a very nominal fee, an anathema to traditional vendors of proprietary software. For example, Red Hat, one of the leading OSS product distributors, package 435 fully-tested OSS products into their distribution for a fee of about $50.

Competitors can (and do) download free copies of the same products and offer them at a lower price. However, the rules of competition are not fully determined by price (which is nominal anyway); rather, companies compete on the basis of the service provided to the consumer, which can only be good news for the latter. More important than sticker-price, OSS allows companies developing and implementing systems to share both the risks and longterm costs associated with a system. By shifting the locus of value from protecting “bits” of code to maximizing the gain from software use and platform development, OSS redefines software as an industry.

Raymond (1999a) identifies the mistaken business and financial models underpinning conventional software development, terming it a service industry operating under the delusion of being a manufacturing one. Certainly, it is a well established fact that the vast proportion of the total cost of software development is incurred in the maintenance phase—with reliable estimates varying from 70% (Boehm 1976) to 80% (Flaatten et al. 1989). This suggests that the model for proprietary software, which operates a high purchase price with a low support fee, does not reflect the reality of the cost distribution in practice. This is recognized in the OSS model where the software is distributed for a nominal fee and companies then compete on service to the consumer. The model views software as a commodity product where the ingredients are free (Young 1999). Brand management becomes critical and customers learn to value a brand they can trust in terms of quality, reliability, and consistency. In these business conditions, OSS companies can learn from the experiences of companies such as Perrier, Ballygowan, and Heinz, where brand image has been successfully exploited in a highly-competitive consumer-driven market.

SOCIO-POLITICAL MICRO-LEVEL AND MACRO-LEVEL (INDIVIDUAL) MOTIVATION.

"Human" motivations for OSS include scratching a developer's "personal itch," the desire for advancement through mentorship, peer reputation, the desire for "meaningful" work, and community oriented idealism.

While technological and economic factors may be sufficient to understand industrial support for OSS, the motivations of individual developers are often socio-political. The nature of the software development craft and those who practice it needs to be considered. These issues will be discussed in detail in the analysis of the who category of the framework below. OSS developers tend to be self-selected and highly-motivated professionals. They may be working on typically long, drawn-out development projects in their own organizations; thus, any of their software output is not usually subject to the prompt feedback of positive reinforcement. When they contribute to OSS development, however, they get a very real "rush" from seeing their code being used and tested immediately (DiBona et al. 1999). Also, the OSS community norms ensure a strict meritocracy where quality speaks for itself. Contributors cannot confer expert status on themselves; rather, it arises through peer recognition. Although now viewed somewhat controversially (Wahba and Bridwell 1976), Maslow's hierarchy of needs, which posits a category of self-actualisation needs (Maslow 1970), has been drawn upon to help explain the motivation behind the committed contribution of OSS developers (Raymond 1998b). Furthermore, the possibility of learning and skill advancement and overt social and political agenda serve as powerful motivators.

OPEN SOURCE SERVER APPLICATIONS

The important open source server applications, which will be discussed in

the following sections include:

- ✓ Infrastructure services
- ✓ Web servers
- ✓ Database servers
- ✓ Mail servers

- ✓ Systems management services

1 INFRASTRUCTURE SERVICES

Infrastructure services consist of basic network services, security services, and file, print, and directory services.

Basic network services include DHCP, DNS, and WINS plus caching services, and routing where that is not done by an appliance. It is typically very inexpensive to provide these services, on the order of $100 per user per year, and this is a commodity activity that any server should be able to perform.

Security services include firewalls, virtual private networking, intrusion detection, antivirus services, authentication, and authorization. These services are difficult to distinguish at times from basic network services and directory services, which support them, or even mail services, such as the case of antivirus and antispam services. Active Directory, for example, provides directory and security services through the same product and the same interface. Sometimes, indeed increasingly often, these are provided by appliances.

Infrastructure Services

A major difference between open source and Windows in this area is that Linux is usually the operating system of dedicated appliances. Security is actually the most common single use of Linux in the enterprise, and this is mostly in appliances. Appliance vendors prefer Linux (or FreeBSD) for two reasons:

They pay no licensing fees.

They can tune the system precisely for their needs.

As a result, these appliances are inexpensive because of the custom footprintand low license fee. Linux networking appliances are also generally very fast. Linux (along with FreeBSD) is generally recognized to have the fastest networking stack, and the code can be further tuned for particular dedicated purposes. Microsoft offers support for appliances also but usually prefers a more integrated approach, where Windows systems run a mix of services on a larger server.

1.1 File and Print Services

In a mixed environment, we will generally use Samba for file and print services. Linux systems also support file sharing very efficiently and easily using NFS and FTP, and this is a good choice in existing UNIX environments.

Another choice is the Novell iFolders technology, which was recently open sourced. Given the current distribution of servers and clients, most organizations are currently using Windows networking, and adopting Samba will be the simplest choice.

Samba allows non-Windows systems to share file and print services with Windows systems. Samba clients function like Windows clients, but for Linux, Mac, or other operating systems, so they see file shares and printers published by Windows or Samba servers. Samba servers function like Windows servers, but on Linux or other systems, so they can publish file and printer shares and also authenticate users in a way similar to a Windows server.

The current version of Samba can authenticate by acting as a Windows NT primary or backup domain controller, by accessing Windows NT domain controllers, or by accessing the Windows 2000 Active Directory.

Samba is an efficient program and scales well. Companies such as Bank of America and Hewlett-Packard use Samba to support many thousands of clients. The program, written by Andrew Trumbull while at SGI, is an implementation of the Windows Networking facility called Server Message Block (SMB). The name Samba is a play on SMB. The protocol traces back to the period when IBM, 3Com, and Microsoft were working together; is also used in OS/2; and is also known as the Common Internet.

File System (CIFS)

We may be able to arrange file sharing within an organization (inside a firewall) by implementing one of a few simple approaches. If information is usually either private or enterprise wide (public), then we don't need a directory system. We can create and share public shares on file sharing systems and teach users to move information for sharing to those shares. On Novell systems, these were usually set up as virtual drives. Once we get beyond public/private into allowing groups or individuals access to specific information, we will probably need a directory of some sort, although not necessarily LDAP. Using Samba, at least since version 3, the choices are:

Use Samba to manage users. Samba can act as a client to an NT server for authentication or work like an NT primary domain server for NT4 replacement, in which case users are migrated from the NT4 server to Samba.

Use Active Directory. Samba can act as a client for authentication to a Windows 2000/2003 Server running Active Directory (but not as a server).

Set up OpenLDAP or another LDAP server and use that as the directory for managing file sharing.

There are several dependent components for Samba. Samba shares printers by using the local printing facility (generally CUPS today on Linux). It also relies on WINS for naming services by default. Originally,

SMB was based on NetBIOS, later on NetBIOS over TCP/IP (with the NetBEUI stack removed). It can now run without NetBIOS, which many organizations require. To see and work with Samba files—for instance, to create file shares or access them—you will need a GUI tool such as Nautilus or Konqueror that supports SMB.

1.2 Directory Services

OpenLDAP is based on the original LDAP server, written at the University of Michigan. It takes a little more work to set up than the commercial alter- 148 7.2 Web Servers natives, but it is open source, solid, scalable, and provides authentication that is configurable for many of the services we will want to use:

Samba file and print sharing

Apache Web server

Courier and Postfix mail servers

The Mozilla browser and other client programs can read user information from OpenLDAP. In addition, we can program access to OpenLDAP from the command line or from our own applications.

2. WEB SERVERS

There are really no other general-purpose open source Web servers to consider than Apache. It has a high share and is the reference standard for a Web server. It is easy to administer and has low overhead, so it works for small sites and systems. The largest Web sites in the world use it. It can be tuned to perform extremely well, and for specific needs. Support for Apache is the gold standard for open source support. The Apache organization is so successful that it has spawned a family of related projects.

2.1 Apache

Apache is by most measures the most successful single open source software project. It is the most commonly used Web server in the world, constituting about two-thirds of all Web servers. A recent Netcraft survey (November 2003) shows Apache with 67 percent of top Web servers and 69 percent of active, against Microsoft's 21 percent and 24 percent, respectively. Active Web servers are usually regarded as the most useful measure of Web server activity, since names reserved but not used are eliminated. Apache has similar shares worldwide across large and small servers including those used for ecommerce. Apache is based on the original Web server written at the National Center for Supercomputing Applications (NCSA) at the University of Illinois in 1993. The first Apache beta was released in 1995.

The name originally stood for "a patchy Web server." Apache runs on many operating systems, including Linux, most versions of UNIX, Windows, and Novell NetWare. Apache is currently available in two series: 2.0.x, which has been available as a production release for two years since early 2002, and 1.3.x. At the time of writing, the 1.3 series is still significantly more used than 2.0, reflecting apparently a conservatism among Apache users. The Apache license allows its inclusion in commercial products, and it is included in IBM WebSphere among others. Apache is structured into a kernel and a number of modules, which includes both statically and dynamically loaded modules supporting extension tools such as Front Page and WebDAV; languages such as PHP, Perl, Python, and Java servlets; and authentication against Samba/NT, LDAP, and various databases. If you are migrating from the Microsoft Web server Internet Information Server (IIS), CGI programs can be migrated without change because Apache and IIS support the same CGI standard. If your Windows programs were developed with ISAPI, ASP, or Cold Fusion, your simplest option is to run Apache on Windows. Programs that use ISAPI require Windows to function, but if you have Cold Fusion or ASP programs and you really want to migrate off IIS, you can purchase modules, from Allaire and Sun, respectively, that allow these products to run on Apache on Linux. For ASP, you can also consider a product called ASP-to-PHP, which does the one-time conversion implied by its name.

Web servers are inexpensive to buy and maintain. Another option is to let Windows and Linux Web servers work side by side for a period. Apache sites install modules to communicate with development languages, typically called mod_X for language X. Over half of Apache sites run mod_php, a little under 20 percent run

mod_perl, and a little over 1 percent run mod_python. Further sites may run programs with CGI. Plainly, PHP is the most common development tool for Apache Web sites. In fact, PHP is the most commonly used language on the Web (Microsoft ASP is second) and its use is growing.

2.2 Other Web Servers

Alternative general Web servers are the commercial products iPlanet (formerly Netscape) server on various operating systems and, of course, Microsoft IIS, on Windows only. There are some niche products in special markets, such as the Red Hat Stronghold Secure Web Server. Some tools or applications—for instance, Plone and Tomcat—come bundled with a Web server, but this is usually as a convenience. They generally allow you to use Apache.

Tux is a kernel-based Web server developed by Red Hat. It is combined with Apache to improve the performance for straight HTTP display. It can improve performance of such pages a lot; in the right circumstances by an order of magnitude or more. This is similar to the caching products offered by IBM and Microsoft. Other Web servers include Zeus and servers included with development products such as Jetty, which is included with Tomcat, but the share of these products is not over 1 percent.

As far as which operating system the Web server runs on, approximately 50 percent of sites run on Windows, 30 percent on Linux, 6 percent on BSD, and 9 percent on UNIX, mostly Solaris, with other or unknown 5 percent, according to Netscape data in 2001. Quite a lot of Apache servers run on Windows

3. DATABASE SERVERS

Most major databases are available on Linux, and have been for years— Oracle since 1998, for instance. The only major modern database that is not sold to run on Linux is SQL Server. The benchmarks and references are there, and the vendors are quite enthusiastic. Running Oracle, DB2, Sybase, CA-Ingres, or Informix on Linux is clearly a safe conservative choice, and any issues or limitations specific to the platform can be discussed with the vendors. This is essentially migration from UNIX to Linux in almost all cases, since the version of DB2 on Linux is the UNIX version. As with any UNIX to Linux migration, switching costs are reasonably low, as access to these databases from other systems is the same.

You can choose to run an open source database, such as MySQL. The open source choice is more likely to deliver significant savings. It is a more adventurous choice than closed code on open source, but there are many organizations already doing this.

3.1 Classes of Database Servers

We will treat desktop servers as a small class of database server, and some elements of desktop systems as client tools. For example, Microsoft Access can be regarded as a desktop application that administers and updates a database server. The server for Access can be an Access database, which can be local or remote, or it can be a SQL Server, upgraded using the wizard provided by Microsoft, built directly using Access tools, or another database accessed with ODBC.

So these types of database products need to be looked at separately:

Online transaction processing (OLTP) servers

Data warehouse servers

Embedded databases

Client access tools including decision support systems

There are open source choices in all of these areas, but some are stronger than others.

3.2 Analysis of Database System Sizes

Research into large transaction processing systems published by Microsoft in 1999 found that, at that time, the following were numbers of transactions per day at the largest commercial organizations processing transactions (not necessarily automated in all cases):

NYSE: 1M

All card and check processing: 20M

Citibank, Bank of America, Wal-Mart: 10–40M

All airline reservations: 220M

AT&T calls worldwide: 200M

Visa did 30M transactions for 400M customers at 250,000 automated teller machines worldwide. That is about as big as it gets. There are a few new technology and ecommerce applications on the Amazon and Google scale that may run higher volumes than these, but most business systems are orders of magnitude smaller.

The TPC Benchmark

TPC stems from a debit-credit benchmark for banking transactions that originated at Bank of America in 1972. The Transaction Processing Council (TPC) was set up to manage an evolving series of benchmarks starting from TP1 in an independent manner. TPC-C, which was introduced in 1992, is the major published transactional benchmark and has evolved to respond to limitations discovered in earlier such benchmarks. The

benchmark supports a mix of five transaction types and requires all elements of the database, such as numbers of customers, to scale along with transaction measurements.

TPC numbers are published with the relevant cost data so price/performance can be considered, and there are clear rules on how cost is calculated. The benchmark is only for hardware and software that can be ordered by customers and is shipping now or will be available within a few months. In reviewing the actions of various competitors, the TPC has learned many methods of enhancing the results by bending the rules. It has met this continual challenge by developing methods to control and eventually prevent this. The TPC is as good an organization for publishing benchmarks measuring business database transaction performance as we have or are likely to have.

Limitations of the TPC-C Benchmark

The TPC-C benchmark is expensive to run. Because of the way it scales, and the precision needed to meet the standards correctly, it takes signifi- cant time and money to run a benchmark (some say $1M). So only a limited number of these are run, depending on the vendors that choose to spend this money. We can only use the data to approximate a solution we are considering, usually by interpolation. Our chosen hardware and software are not likely to have been specifically tested, and we will look for something similar.

The cost also means that running our own TPC-C benchmark is almost certainly prohibitive, but it is generally desirable to do this. Another method is needed to allow us to get really specific in addressing our needs. TPC cannot enforce that the methods used for the benchmark are the methods actually used in the real world. One reason the benchmark is expensive is that the skills to set it up are unusual, because it is now usually run on quite specialized software that ordinary organizations would not use, as follows:

Most vendors use custom C++ code and the Tuxedo application server, while recommending Java application servers.

Big database measurements use tricks such as distributed partitioned views, which customers don't like to maintain, and materialized views, which customers do not benefit from.

TPC-C prohibits methods such as queuing that allow smaller databases to manage high-peak workloads.

The Winter Top Ten Lists

Winter Corporation publishes Top Ten lists of large production databases, both OLTP and DSS. The statistics are self-reported by customers, sponsored by database vendors, so it is a little bit of a "bragging contest." There may be larger systems that choose to remain anonymous.

However, the data reported appears to be accurate and includes some of the largest systems, so it is very useful for my purposes here, which is to get a sense of how big databases really are and what platforms people really use.

In particular, the real-world Winter measurements, like the real-world research numbers, are much lower transaction rates than the TPC-C high performers. So this confirms that the best (and most expensive) systems being measured today are substantially outperforming the requirement. So we have a theoretical measurement and some practical measurements.

In the largest production systems, we actually see only three platform combinations: IBM mainframes, SQL Server on Windows 2000, and Oracle on UNIX. In the TPC data, we see Oracle on Linux with a couple of very high numbers. This is an important breakthrough for Linux, which is in the TPC measurement for the first time, but not in production.

3.3 Open Source Databases choice

There are three open source databases to consider seriously for general use, in my view. These are Berkeley DB, PostgreSQL, and MySQL. They are all widely used.

Two more systems play in niches. MaxDB was formerly known as SAP DB and before that Adabas-G or Supra. It has some major clients, mostly in Europe, but has never caught on in the United States even with the pull of its integration with SAP. Its role will probably be to bring technologies for incorporation into future large-scale versions of MySQL.

Berkeley DB

Berkeley DB is a core tool under several important open source products, and apparently has 200 million deployments. Berkeley DB (BDB) is a highperformance derivative of the old "DBM" databases, which have been part of UNIX and UNIX-like operating systems from the beginning.

As an embedded, or application-specific, database, BDB is included with products, often without the user being aware. This is like Btrieve or Microsoft Jet and MSDE engines. It is a flat-file database, not SQL.

BDB has a dual licensing model. It is open source (GPL license) when used in open source products or at a single site. When distributed with a commercial product, there is a commercial license.

Berkeley DB is used by Sendmail, Apache, and OpenLDAP servers, the Netscape and Mozilla browsers, and the Python and Perl programming languages. Commercial customers include Sun, Google, Veritas, TIBCO, Cisco,Amazon,and HP.

PostgreSQL

PostgreSQL is an open source database, available under the BSD license/ copyright regime. It is based on the Postgres product designed at Berkeley in the 1980s, and before that on work performed on the Ingres database by Michael Stonebraker since 1974. It was not a SQL-based product until 1995. Ingres and Postgres were developed on BSD UNIX. It is available now on Linux and UNIX, including the Mac, but is not native on Windows, running in the Cygwin emulation.

Postgres has historically offered better support than MySQL for standard SQL behavior, although MySQL seems to be catching up. Currently,

MySQL

The MySQL database server is robust, fast, and a very good cross-platform product on clients, including Windows and the Mac and a variety of UNIX servers. It has a small footprint and good management tools. The product is distributed by the Swedish company, MySQL AB. MySQL is used much more than PostgreSQL; the company estimates about 4M users worldwide. The product has momentum, with considerable enhancement happening, and last year's acquisition of MaxDB will likely lead to more enterprisescale features later.

MySQL is dual-licensed, meaning it is available under a commercial license or the GPL. Because linking with the GPL-based libraries requires your code to go GPL, commercial developers who are not open source will want to pay for the commercial license. Also, the MySQL company asks commercial users to buy an unlimited commercial license. That commercial license is $500 for the product, including InnoDB, which is transactional—that is, unlimited processors and users. With this licensing model, MySQL is powerful and inexpensive for commercial users and it is open source for government, education, and personal users. Historically, MySQL has missed some SQL standard features that many users regard as essential. This was originally a set of design decisions, as the product was intended to be fast above all. There is now a published plan to catch up on these, which is in progress. Transactions (ACID) were released in Version 4.0, when the previously distinct InnoDB engine was incorporated in the main product. Subqueries are in Version 4.1, which is close to production as I write. Stored procedures will be in Version 5.0 and triggers in 5.1.

MySQL is powerful enough for most purposes and easy to install and

use. It is widely used in business organizations, including large systems such as Sabre. It powers the OSDN sites, including Slashdot, Freshmeat, and SourceForge, and is used by Google and Yahoo!.

3.4 Database performance is good enough

In mid-1999, an eight-way Microsoft SQL Server 7.0 reached 40,000 tpmC on one server with a then five-year cost of $.75 million. Because of the way the TPC-C benchmark is scaled, the 40,000 transactions in August 1999 represent 90M customers, 300M stock items, 120M transactions per day, 32,000 simultaneous users, and 5 terabytes of storage. At that time, SQL Server was good enough by transactional measures for almost all actual business database uses.
Improvement has continued at this pace. By 2002, the fastest SQL Server benchmark was ten times quicker than that. Price/performance on many typical systems is now below $2/tpmC, which is ten times better. Put another way, the tenfold improvement of the last two and a half years can be taken as better performance or lower price.
Databases are used as a component in a complex system. Most databases in organizations sit behind Web sites. Others are behind client/server applications or distributed. Their performance is constrained by the front-end systems and the end-user needs. Most are in the medium or large categories, but not very large like the TPC-C record breakers.
During these years, Oracle has usually held the highest performance benchmark, with SQL Server catching up periodically.

3.5 Competing with Closed Code Databases

You cannot install a large database in any organization without having to compete with sales pitches from Oracle, Microsoft, and IBM. Salespeople from any of these companies will try to treat open source databases as toys. If they are forced to admit that the open source database could do the job under discussion, they know they will lose on price, so they will move the debate elsewhere. They will talk about theoretical large databases and grid computing, their results at the TPC-C racetrack, or the idea of consolidating all your databases into one big system.

4. MAIL SERVERS:

In the UNIX and open source world, mail servers are split between message transfer agents (MTA), which are senders, and receivers/message stores. Mail is usually sent with the SMTP protocol and accessed with the POP3 or IMAP protocol. This works about the same with closed code mail servers such as Exchange, but both sending and receiving programs are called Exchange (or Groupwise or Notes).
We have already mentioned Sendmail, the venerable program that may be the oldest open source program in widespread use. After paying it due respect, it is time to admit that Sendmail is an old program and may not be the best mail server to choose today. It has a reputation for being difficult to configure and a history of security problems. The consensus these days is that you should choose Postfix instead to avoid these issues. There are other alternatives, such as Exim and Qmail, but we will look at Postfix here. Postfix is fast, scales well, and is reasonably self-evident to configure. It can use different formats for the message store (Maildir or Mbox). We usually prefer the Maildir format, which stores each message in a single file. This makes message processing with external tools much simpler.
The alternatives for message receivers and stores are POP3 and IMAP. IMAP is richer and generally preferable. Exchange supports either protocol. The native Exchange store appears to be IMAP-like but differs slightly, so that Outlook IMAP support can be quirky. Sometimes we may choose to use POP3 with Outlook for that reason. Choices for IMAP servers include Courier-IMAP and Cyrus IMAPD.
For a directory server, we prefer OpenLDAP for this. Postfix and CourierIMAP or Cyrus IMAPD can access OpenLDAP for authentication.
Many organizations like to have a browser-based mail client option. Horde is an example of a server that supports browser-based email. Horde looks similar to Outlook Web Access and provides similar functions. It can access OpenLDAP for authentication, address lookups, and contacts.

5. SYSTEM MANAGEMENT

The basic choice for open source systems management, as in other areas such as database, is whether to adopt open source tools and methods completely, which will involve getting or developing administrators with UNIX administration skill sets, or whether to adopt closed code system management tools, which are generally cross-platform and may already be in place in the organization. Both approaches will probably be needed. However attractive the

graphical tools, system administrators usually need a thorough understanding of the platforms they are using.

The next level, if needed, of systems administration is programming using a cross-platform scripting language. This will allow us to develop more flexible and automated approaches to, for instance, backing up or managing the size of user files. The good news is that there is a long tradition of scripting in UNIX, and the work to do this is well understood and available. The bad news is that although it is compatible with Windows systems, it is not compatible with the approaches that have generally been used in Windows.

The closed code tools are comprehensive and graphical, so they look wonderful in use. The open source tools are generally targeted to a more experienced administrator, and lean more to the UNIX philosophy of "doing one thing well." There is no reason not to use tools of both types. Many organizations that use Tivoli or Unicenter also employ open source tools such as Snort for intrusion detection and write shell or Perl scripts to manage their own applications.

The leading integrated graphical open source system monitoring tool is Nagios. This is being used in production by organizations with up to 5,000 hosts.

Open source administration tools include a huge selection of specific tools for particular purposes. Most existing larger organizations will have a multiplatform administration solution in place and will simply extend it to include the open source systems. Systems need to be monitored at all levels. Open source applications are easier to instrument to support event logging into system management tools. Other great open source tools include TCPdump, Snort, and Ethereal.

OPEN SOURCE DESKTOP APPLICATIONS:

1. INTRODRUCTION

1.1 THE OPEN SOURCE DESKTOP

A complete open source desktop with applications can be easily installed and demonstrated on a typical personal computer using Linux. Most people would agree that such desktops are attractive, powerful, and as easy to learn from scratch as Windows. Such desktops can be significantly less expensive than closed code systems, since they can save the operating system cost plus the cost of applications such as Microsoft Office.

It is also possible to build a desktop on Windows, where all the applications are open source. Again, this can be attractive, powerful, and easy to learn. In many situations where the operating system is already installed, such as on home computers, there is no savings to replace it with Linux, but there are huge cost savings from replacing applications such as Office with open source.

The important open source desktop applications, which will be discussed in turn, are:

Graphical desktops
Web browsers
Office programs (word processing, spreadsheet, presentation software)
Professional applications (graphics, database front ends, Web designers)
Personal applications (media players, games)

1.2 LINUX DESKTOP SHARE

Linux has come a long way in power and ease of use, but it is still not widely used on the desktop. Linux has now overtaken since 1994 the Mac in sales to become the #2 operating system on the desktop. IDC reports that Linux grew from 2.8 percent in 2002 to 3.2 percent in 2003, while the Mac remained at 3 percent.

This is significant, but is still has only a small share. Windows has a 94 percent share. IDC forecasts growth to 6 percent for Linux in 2007, but Windows would still be over 90 percent by then. These figures probably undercount Linux presence on desktops now and in the future. Linux is underreported, because it is very often not purchased. Windows ships on essentially every new PC, and where users are replacing Windows with Linux they are probably not getting measured effectively. It is also used in concentrated niches, some of which are very high growth, such as some new Asian installations involving millions of desktops. The major computer companies—IBM, HP, and Sun—all have programs to encourage Linux desktop adoption now, and some major corporate announcements have been made. Linux has exceeded expectations in the past, and may grow on the desktop much faster than currently predicted.

1.3 LIMITATIONS TO DESKTOP LINUX ADOPTION

Whatever Linux growth may be, in the next three or four years we know that there will continue to be an order of magnitude more Windows users than Linux users. This has an effect on the availability of hardware, applications, and support

services. Each of these limits the possibilities of Linux desktop deployment significantly.

Hardware

Approximately half of corporate personal computers are now notebooks rather than desktops. It is this group that is least likely to adopt Linux quickly. Setting up Linux on a notebook system is still likely to need some custom work, and there are hardware limitations, including wireless support, such as Intel Centrino wireless and most 802.11g cards; some graphics cards; and advanced power management. Notebook users may have to accept some loss of functionality to run Linux. Notebook users are typically professional users, and are not likely to accept compromises like this unless they are developers or are committed to open source for some other reason.

Applications

Given the disparity in installation share, it is perhaps surprising that there are many applications available for Linux desktops, and there are generally several good choices in the major categories. Smaller niche applications are more of a problem. There are many thousands of applications in the Windows "ecosystem," usually written to the Windows tools and interfaces, often addressing specific vertical industries.

In a migration situation, any specific application may be a "must-have" for a group of users Microsoft Office is just the biggest, best-known example of this. Section 8.6.1 has some tactics for this situation, such as emulation, but often this will necessitate Windows on some systems.

SupportServices

There are thousands of corporate employees, and many more people in outsourced services, working with users of Windows desktops in support and training roles. Some of these have qualifications such as MCSEs, others do not; but most have a significant investment in the skills needed to support Windows systems and the common applications deployed on them. There is little incentive for these people to relearn their jobs using a new technology, and in some areas the skills to support activities such as solving issues with Linux systems that won't boot or training users in OpenOffice may not be available yet. It will be several years until this situation is resolved entirely.

2.GRAPHICAL DESKTOPS

It is possible to start and run Linux in a character mode, but this is reserved for installation and debugging situations nowadays. Linux is usually installed to start in a graphical mode, running a windowing system, and a desktop system will usually run a desktop manager with a set of integrated utilities. If you want to work in the shell using a command line and typing commands, as you probably did sometimes in Windows, you can open a terminal. Linux users tend to use the command line more than Windows users, partly because it is more powerful. Essentially all Linux systems use the X Windows system (X11) as the graphical user interface (GUI)—generally XFree86, which is the most used port of X11 on Intel. This is the underlying code of Linux graphical user interface systems. Exceptions include some servers that may not need a GUI and run in character mode, and some embedded systems that use other GUI systems not based on X11 to get better performance, such as Qtopia, which is used on the Sharp Zaurus.

3.WEB BROWSERS

When developing applications, we do not usually want to require a particular Web browser and operating system. In many cases, we cannot know which browser an application user will be using. Even if we can determine this, as in a customer or business partner situation, it is probably an unreasonable restriction to impose. So when developing, we will usually plan to support a choice of browsers.

3.1 Deploying Browsers

Although we may want to support several browsers when developing, when deploying desktops we will probably want to use a single standard to lower the support and training burden. Most Windows shops use Internet Explorer (IE) for obvious reasons: It is good enough and is already installed. The limitations of IE, such as its lack of control over pop-ups, can be addressed with third-party add-ins or managed from the firewall.

Organizations that would like a single browser across multiple platforms can select the open source Mozilla, either Firefox or the older integrated versions, Netscape or Opera. The other browsers are specific to their platforms: Safari on

the Mac, Konqueror on KDE, and Epiphany and Galeon on Gnome.

4 .The OFFICE SUITE

In considering the office suite, we will consider the word processing, spreadsheet, and presentation programs, although the open source suites, such as Microsoft, include other programs, such as drawing and image management. In terms of Microsoft Office, then, we are looking at replacements for Word, Excel, and PowerPoint. We will discuss database front-end programs and mail front-end programs such as Access and Outlook elsewhere under separate headings. There are excellent open source equivalents of these and of other Office programs, such as drawing, organizational charts, spell checking, and so on, but the core components are usually seen as these three.

There are several alternative open source office suites:

OpenOffice
KOffice
Gnome Office

KOffice and Gnome Office contain some good products, and many individuals may find them to be exactly what they need, particularly when working with other programs from those desktops (KDE and Gnome). But OpenOffice is clearly the strongest. It has three very powerful constituent programs, and is the best office suite for working with Microsoft formats. OpenOffice has a great deal of momentum, with millions of users, far more than the others. OpenOffice has been adopted as part of the standard desktops of Red Hat, SuSE, Ximian, Sun, and UserLinux. OpenOffice works well on Windows and Linux. Anyone recommending an office suite as a standard to an organization really has to recommend OpenOffice. An alternative might be not using a suite, but allowing individual programs to be selected.

4.1 OpenOffice.org

OpenOffice.org (abbreviated here to OpenOffice) is the leading open source office suite. Sun purchased StarDivision, the German developers of StarOffice, in 1999, then established OpenOffice.org to manage the open source process and distribution while continuing to offer StarOffice on a commercial basis. StarOffice and OpenOffice share the same code base and file formats; OpenOffice is open source while StarOffice is sold commercially and contains additional

features.

At the time of writing, OpenOffice is at Version 1.1 and StarOffice is at 7.0. The main programs are identical, but StarOffice includes additional products in the distribution, including TrueType fonts, spell checking and thesaurus utilities, additional templates and pictures, and a desktop version of the Adabas database, called Base. Sun also offers commercial support for StarOffice. In this book, we will from now on refer to OpenOffice to include StarOffice as a possible choice.

Fonts and spell checking are weak in OpenOffice as shipped. An organization adopting OpenOffice should look at options for these functions.

Another example of OpenOffice integration is Ximian. The Ximian edition of OpenOffice makes changes to ease Office migration, using Microsoft file formats by default and shipping Microsoft-compatible fonts. It also makes changes to improve integration with the Gnome programs Galeon and Evolution and to recognize Gnome desktop theme and font settings.

File formats are identical between OpenOffice and StarOffice and with the previous versions (1.0 and 6.0). OpenOffice 1.1 is available for Linux and Windows. These versions are essentially identical. The Mac OS X version of OpenOffice is 1.0 as I write. File sharing is still possible, but some functions of the program are a level back. The Mac version is not native but based on X11. Between the back level and the nonnative issues, I have found the Mac version of OpenOffice to be too slow and with a poor onscreen format. This will be fixed in a few months. At the moment, there is a version called NeoOffice/J, using a Java front end, that is fast and presents very well.

4.2 Competition in the Office Suite Market

Microsoft Office appears absolutely dominant in its market, with market share over 90 percent among office suite customers. This may seem impossible to tackle, but there are other ways to look at this.

Since there is over a 40 percent piracy rate claimed for Microsoft Office, for every three licensed users there are two more who did not buy it. There are also millions of people who got free copies of the Lotus suite with their IBM systems, a few million OpenOffice users, and the 10 percent who bought some other suite.

Finally, there are also many people who do not use an office suite, but use individual programs or simpler packages such as Microsoft Works, or

use text editors and Acrobat. Many of these might like to use an office suite but cannot afford Microsoft Office.

Problems with Licensing

Concurrent licensing is a scheme where use is tracked and the organization pays for use, generally for the "high-water mark" of use. The idea is theoretically attractive to a customer organization; after all, of a few thousand Excel users, how many are actually using it at any one time, particularly if you make readers available? It is very unattractive to the selling company. As the cost of a concurrent license is pushed higher, some users balk at purchases, since the price seems excessive, although even at 10 or 50 times, it still brings less revenue to the vendor than licensing everyone. The information on use that is essential to a concurrent scheme provides feedback that can be used to lower use further—for instance, by spreading out a period of peak use. But that issue cuts through to the problem at the heart of Microsoft Office. Most people don't use most of it.

4.3 Comparison of Microsoft Office to OpenOffice

Bundling

OpenOffice does not include an email client like Outlook, but most people will use Evolution, which is powerful, similar to Outlook, integrates well with OpenOffice, and is open source. Similarly, OpenOffice does not include a database program, but most people will consider MySQL if they need a SQL database program. MySQL is more powerful and scalable as a database than Access, but has no equivalent integrated front end. Possible front ends include Mergeant, the Gnome database front end, OpenOffice, and database tools such as MySQL administrator and Quest.

Integration

OpenOffice can connect to databases using the access methods ADO, JDBC, or ODBC. It can for instance connect to Access using ADO, MySQL using JDBC, and SQL Server using ODBC. OpenOffice formats are XML based and published, so integration with other systems is simpler than for Microsoft Office. It is not necessary with OpenOffice to buy more expensive editions to manage XML formatting. There is a software development kit for extending OpenOffice using Java.

Formats

There are some serious problems with using the Microsoft office formats. They are proprietary, subject to change, and not documented. It is quite

difficult for third parties to access them, although some good programs are available. Using these formats in correspondence, for instance, is implicitly requiring others to acquire the Office programs when they may not own them or need them.
OpenOffice uses a zipped set of XML files. In practical use, the OpenOffice format is generally no more than half the size of the MS Office files (unless bit maps or other uncompressible attachments dominate the size).
The OpenOffice format is also simple to read using standard tools, because the XML format is published.

4.4 Migration from Microsoft Office to OpenOffice

The Microsoft installed base is Office 97 and Office 2000. There is very little Office XP or Office 2003 yet. This immediately highlights the main problem with Office migration today, which is that nobody wants to do it.

Importing/Exporting between MS Office and OpenOffice

First off, Microsoft Office cannot read OpenOffice files at all. Any OpenOffice files must be converted to Microsoft formats in OpenOffice.
OpenOffice-specific features will be lost in these formats.
OpenOffice is very good at reading Microsoft Office formats, but not perfect. There are several formats to consider, such as Office 95, 97, XP (2002), or 2003. Microsoft Office since Office 97 uses an OLE format of structured storage, typically containing several "streams" of information.
Office 97, 2000, and XP use the same formats (for these three programs).
The earlier Office versions, 4.2 and 95, used different incompatible formats, but are unlikely to be met in corporate environments today, partly for that reason. Every version has a different macro language—for example, WordBASIC, VBA 5, VBA 6, and VBA 6.3.

4.5 Lock-in and Complexity

Not all organizations will be able to migrate away from Office now. It depends on the way they use it. Users will be slower to change if they are locked in, because they use Microsoft Office features that do not migrate.
How seriously your organization is locked in needs to be evaluated for each group of users in the organization. It is a function of:

Advanced or professional users authoring documents
Use of technical features (macros, shared components, etc.)
The overall pattern of collaboration over documents

One-Time Migration

If you plan a one-time migration versus continuing interoperability, you can handle most of the problems in a reasonable way. If your intention is to stop using Office and migrate all documents to OpenOffice, you will find that most documents will transfer with minor format changes that will not bother users on a one-time basis. Hundreds of documents may be transferred with a few hours clean-up work. For example, page numbering may be slightly off, some fonts may be replaced in a way you don't like, and some complex references to external files may need to be checked. In this situation you can test as many documents as you need, and also arrange for some expertise to be available to support the migration.

Two-Way Interoperability

If you intend to continue transferring documents back and forth on a daily basis, the hours spent "fixing up" documents will add up indefinitely and become an impossible burden. That is why patterns of use need to be analyzed. Any plan that involves a regular exchange of complex documents back and forth between the different office products will need to be reviewed carefully, and preferably altered to eliminate this.

The Effect of Switching Costs

OpenOffice is a good product that meets the needs of most poeple in most organizations for an office suite. It is good enough. Most people can install OpenOffice and gain a system that does everything they need. However, most organizations already have Microsoft Office in place, and that changes everything. Unlike server products where switching is easy or even undetectable, changing the desktop is a big deal.

4.6 When You Don't Need an Office Suite

Individuals and organizations that don't need to pick a suite can look at individual products such as the AbiWord word processor and the Gnumeric spreadsheet as possible "best of breed" choices. The idea of an office suite for everyone is a relatively new idea and not particularly natural. Originally, PowerPoint was used by marketing and sales departments, and spreadsheets were used by accounting. Many corporate writers only need an email program. In a company that licenses thousands of copies of Office, it could be that there are only a few hundred (or a few dozen) who create original Word, Excel, or PowerPoint documents. Many of these use only a fraction of the available functions.

A real-world example of a successful system that is generally used effectively without an office suite is the Apple Mac. The Mac as sold includes TextEdit (which can read/write Word documents), AppleWorks (which can read/write Excel documents), and Mail/iCal/Address Book, which work together, similar to Outlook or Evolution, and serve as a client to Microsoft Exchange. Keynote, which can read and write PowerPoint documents, is sold separately. Apple applications are well integrated and consistent without being a "suite." Apple users can choose to buy Microsoft Office, or can install OpenOffice, but most do neither.

5. MAIL AND CALENDAR CLIENTS

There are several good email open source clients available. This includes browsers that also do mail, such as Mozilla and Opera, and dedicated email clients, such as Eudora.

A big question with mail programs is the extent to which you want to replicate Outlook. If you want the Outlook features, including the bundling of calendar, email, and small-scale personal databases, you will probably want to use Evolution, which matches the look of Outlook very well.

Evolution includes mail, calendar, task list, and contacts and offers screens that combine all these. You can use Evolution as a front end to Microsoft Exchange, using a connector available (for a fee) from Novell/Ximian, or use Evolution with other mail servers that support POP, IMAP, or MAPI.

5.1 Professional Applications

This includes applications for project management, drawing and image management, and other professional work. In some ways the situation is similar to that with Office. There are good open source programs available, but they may not match feature for feature, and migration raises problems of data formats and user training. An application inventory is going to be necessary.

5.2 Drawing and Image Management

The open source programs GIMP, Dia, and Sodipodi compare favorably for general users with PhotoShop, Visio, and Illustrator. As with Office, the most demanding professional users will not switch because of their time invested and high-end neeeds. Most people will find these programs more than sufficient. GIMP is available on Windows also and is a very good image program for professional and home use on that platform as well as Linux. Dia is similar to Microsoft Visio. It does not have a comparable array

of stencils available, but it is good enough immediately for simple work, and for custom work the format for creating shapes is open and reasonably easy to use. Most of the diagrams in this book are created with Dia.

6. PERSONAL SOFTWARE

This area is one where Linux is catching up quickly. I still think that today if you simply want to choose the best machine for performing multimedia functions or games, no other considerations, you should look at the Macintosh or Windows XP. Mac OS X is the leader in graphical user interface and multimedia tools, particularly tools that are integrated and easy to use, while Windows XP is the leader in PC gaming, with far more games available and specialized hardware, which is easy to install and support. The Sony Playstation is another good gaming choice. Linux cannot match the PC or Playstation for variety and currency of games.

6.1 Running Windows Applications

Sometimes we have to run an application that is not available on Linux. Most needs can be met in a general way, but there are quite often particular programs that are not available. If it is necessary to run a particular program that is not available on Linux, this can be met with a variety of techniques. First, we can check against a Web site such as the table of Windows equivalents at http://linuxshop.ru/linuxbegin/win-lin-soft-en/table.shtml to see if there is a Linux equivalent. If there is not, and we cannot match it or migrate it, we can host Windows programs on Linux using the emulation program Wine. This program is also packaged with additional material as CrossoverOffice to run Microsoft Office. Using Conexant drivers, we can access hardware that requires Windows drivers. With VMware, we can even run a complete Windows operating system on Linux. Of course, these options are not inexpensive, since they involve the emulator and the licensed Windows programs.

COST OF OPEN SOURCE SOFTWARE

There are not always similar products and where there are, we may have a preference for one feature set over another. we will only compare costs.

Then we will examine the total cost of ownership of open source and closed code products and compare those. To do this, we will take some scenarios for businesses of

different sizes. The estimates for hardware and staffing are kept very simple, and you are invited to substitute your own numbers.

Because so many factors differ, this is only a framework, which will need to be adjusted for a particular organization. The tables here are available at the Web site *www.kavana.org/opensource* for download if you would like to adjust them for your own situation.
We will review these costs by category, and then put the per unit prices into simple tables. We can use these tables as the cost basis for some typical scenarios. In all cases, we should substitute our local information into this table, since our prices may vary.

1.Total Cost of Ownership

There is a simple answer to the question of open source software costs, where open source solutions are comparable to closed code alternatives. When compared with similar closed code systems, open source systems as a general rule cost:

Much less for software
No more and often less for hardware
If other things are equal, no more for anything else

As far as software costs are concerned, we will review tables with the prices for common open source and closed code products, and see that open source software costs much less.

As far as hardware is concerned, open source products are available for effectively all current hardware platforms, including the systems with the best price/performance. Open source performance on a platform is usually similar to closed code competitors, as already discussed throughout this book. So hardware for open source software generally costs the same as for the least expensive system for closed code. In most cases, we are comparing the same hardware running Windows or UNIX on the one hand versus Linux on the other.

Other things may not be equal and total cost of ownership (TCO) studies offer an opportunity to show that. There are many forms of these, and there is a small industry that compares and contrasts TCO versus ROI versus various other terms. Here, we will keep this simple and use TCO to

include the other costs involved over a reasonable period of time when making a software decision.

The issue usually comes down to staffing costs. There are some published TCO studies that attempt to show that open source software costs more than you think, or that hardware costs more for Linux in some specific situation, but they are from obviously biased sources and are not really credible. The three big cost elements of TCO are staffing, hardware, and software. Of these, staffing dwarfs the others in all the scenarios we will look at. Because of the dominance of staffing costs, even where open source software saves millions, this will not represent a particularly large percentage difference in TCO. However, software may be the only controllable cost. In these cases, TCO can obscure the real savings by adding large costs, which are effectively fixed, such as system administration and support, to both sides of a comparison.

1.1STAFFING COSTS

Personnel costs dominate software costs for infrastructure. Because of this, the savings from open source software such as Linux and MySQL will be small compared with the costs of personnel for development and management.

An IDC report on Windows and Linux infrastructure costs estimates the TCO cost breakdown for infrastructure.

This may understate software costs, but it is broadly consistent with work by Gartner on IT costs, which again shows staffing and downtime as the major costs for infrastructure. So for IT infrastructure systems, the impact of a system on system administration and end users can be ten times more important than its purchase cost. This indicates how inexpensive IT infrastructure is today measured at the server.

Application solutions can be much more expensive. Large applications can incur millions of dollars in costs for software acquisition or development, as well as large server hardware costs, particularly for database systems. Even for simple Web applications, hardware and software are higher than for infrastructure.

The costs are loaded, including salary, vacation, management overhead, general training, taxes, and benefits. They are averaged, with no effort to distinguish between skill levels. There is also an entry representing a week of training. Many projects require training of a week or two for developers and administrators.

Support costs are difficult to calculate, because there are so many different options. For mission-critical systems, an organization will want to contract with the system developers to ensure coverage whenever there is a system problem. For desktop systems and infrastructure, it is usually enough to maintain competent staff and solve the issues in house. A support contract with a software vendor such as Microsoft Product Support Services, providing a full-time equivalent, costs upward of $250,000. Contracts involving a named contact and some number of incidents might start at around $50,000 annually. Similar contracts can be struck for open source software products. They are likely to cost much less (a third or a quarter as much) and will be structured less formally.

Staffing Costs

Item Cost	Per	Details/Comments
Developer	$95,000	Year Loaded cost
Sysadmin	$75,000	Year Loaded cost
Training	$10,000	Week Including class, travel, and expenses

Item Cost	Per	Details/Comments
Big 4-processor box	$25,000	Server HP DL745 4-processor 2gig RAM
Medium box	$12,000	Server Dell 2650 2-processor
Small box	$4,000	Server Dell 1750 2-processor
SAN, shared disks	$80,000	Project Dell/EMC

1.2 HARDWARE COSTS

Hardware costs include servers, clients, networking equipment, and other appliances such as firewalls. Hardware costs are generally about the same between Windows and Linux, unless there is some unusual performance issue causing a difference. Usually, the same hardware can be used at all levels. Before the recent releases of the Linux 2.6 kernel or some specialized late 2.4 kernels, such as Red Hat Enterprise Linux 3, Linux threading was slower than the hardware allowed, and this had a negative effect on database and application server measures.

Table 12.2 includes estimates for hardware. Most organizations will use a few standard boxes so that service parts can be stockpiled and one set of trained users can maintain all the systems. I have used commodity systems of the type commonly used for Windows and Linux. Most systems can be put together with these components. There are, of course, much more expensive servers available for specialized purposes.

1.3 SOFTWARE COSTS

The closed code comparative software prices that follow are given for Microsoft. The major vendors track each other's pricing and Microsoft's pricing is more transparent than other larger vendors. In my experience, Microsoft is very rarely more expensive for the same class of product than IBM or Oracle, and its pricing is relatively stable, easy to get, and easy to work with.

1.3.1***Infrastructure Software***

we will include the operating system and any essential tools for networking and system management. There is a variety of good open source administration tools available. Windows includes directory, file, and print services; simple routing; and a Web server, so we will count them in also. In Windows environments, firewall and proxy services (ISA) and mail (Exchange) are additional products, and we will factor that into the costs since organizations generally need those services. There is a simple mail server included with Windows Server, but this is not usually used for enterprise mail.

The majority of Windows customers do not use the more expensive server products such as BizTalk Server, Content Management Server, Sharepoint Portal Server, or Commerce Server. These products cost from $10,000 to $40,000 per processor. Competitors such as WebTrends, Vignette, Plumtree Portal, or Blue Martini are even more expensive. We will do one comparison using these types of products for completeness.

Open source software will generally be less expensive. In addition, license tracking is wholly or partially eliminated. For Windows, client access licenses (CALs) must be counted for all these, including directory access. Client access licenses are generally the largest software cost element.

1.3.2 ***Database and Development Software***

In a Windows environment, this usually includes SQL Server and Visual,Studio as items of additional cost. Other development tools, such as the IIS Web server, the .Net development framework, application server components, and Active Server Pages, are included with Windows Server. In an open source environment, we will take this to include MySQL, Apache, and PHP, and in some cases JBoss and Tomcat. These products generally ship with and always install on popular enterprise Linux choices, such as Red Hat and SuSE. MySQL has a small license fee in a

commercial environment, and JBoss has a support charge; we will include those where appropriate.
Many organizations will use Oracle or DB2 as the database. These typically cost as much or more than SQL Server.

1.4 USING THIRD-PARTY APPLICATION AND DATABASE SERVERS

In both the open source and Windows environments, there are many alternative choices of third-party tools and database servers. Popular choices include:

Oracle or IBM DB2 database servers IBM WebSphere, BEA WebLogic, or Oracle application servers Tools for modeling, debugging, code management, and so on, such as Rational and ClearCase

These products have the same performance and functionality and are about the same price in either environment. People who choose these products generally choose them at least partly for this ability to offer the same experience across the Windows and Linux platforms; they do not see the Windows-only tools as equivalents.
These products are very expensive in comparison with open source software or Windows development software. In 2003, for example, IBM was listing the following prices:

WebSphere Advanced Server	$11,400	per processor
WebSphere MQ	$5,000	per processor
WebSphere Interchange Server	$123,000	per processor

Counting the necessary maintenance and support contract, the threeyear price is twice that quoted. So the list price to put WebSphere on a couple of four-processor servers to perform a typical complex Web application with components and queuing will cost $32,800 □□8, which is $262,400, not including any database. There are several warnings to consider with this price; there are lighter, less expensive versions of WebSphere that will work for many situations, these prices are subject to discount, and may have changed since the time of writing.
The effect on cost calculations of including these products is to add a fixed (large) element to each side of the comparison, damping the overall difference. Of course, if you add these products to one side of the comparison only, their cost will determine the outcome, but your comparison will be of very limited value.

1.5 PRICING OPEN SOURCE SOFTWARE

Table lists the prices of commonly used open source software products. Note that you can always distribute an open source software product, so you only need to buy a single copy to get documentation and CDs. The two exceptions here are MySQL, which is sold under a commercial license priced per server, and Red Hat Enterprise Linux, which is only sold including support, so that is also priced per server.
All prices for open source software are per system. Note that the RedHat Linux product prices include support. The MySQL database is duallicensed, with a commercial license price of $500 per server, or is availableas free software under the GPL; we included it as $500.

Open Source Software Prices

Product	Price	Function
Server Software		
Fedora Core	$0	Server OS
Debian GNU/Linux	$0	Server OS
Red Hat WS Standard	$300	Server OS
SuSE Standard Server	$450	Server OS
Red Hat Enterprise Linux AS	$1,500	Server OS
SuSE Enterprise Server	$1,000	Server OS
Squid and iptables	$0	Proxy and caching
OpenLDAP	$0	Directory
Samba	$0	File and print sharing
MySQL Commercial	$500	Database

1.6 PRICING CLOSED CODE SOFTWARE

It is difficult to fully determine closed code software costs for several reasons. Not all systems have a published price list, and the lists that exist are incomplete. Products are often offered with very different prices to different customers, and even different pricing models. Most companies offer substantial discounts, which are not published, to large customers. Some products are only available through personal contact and quotation from a salesperson. Prices can change substantially overnight, such as Oracle database prices, which went down with the release of 10g. Complex products such as WebSphere have many components and several different pricing

models. Deals can be made, particularly if vendors know they are in competition with lower-priced products.

Although it is not possible to predict the precise number that a vendor will quote to a particular customer, we can get a good estimate of the selling price for that class of customer. For example, I have quotes received in recent consulting engagements, or shared with me by customers, and all companies provide pricing examples for their published benchmarks and comparisons.

When we calculate prices in detail, many things can raise prices above the initial expectation. Two examples are add-on products and software maintenance. These costs are usually higher for closed code. Software maintenance is commonly 25 percent of the purchase price annually.

1.7 PRICING WINDOWS SOFTWARE

Microsoft has a published price list, so we can work with those numbers.

Table lists prices of Windows software. It is often a good practice to compare list prices, since discounts are unpublished and can vary considerably. Although list price comparison usually tends to be roughly fair, it is not fair when comparing open source with closed code. Closed code software has higher prices and is often discounted considerably, so ignoring discounts will tend to overcount the price of the closed code. I have used list prices in Table 12.4, because that is what is available publicly. Large organizations should often be able to get substantial discounts—for instance, 25 percent less than these prices.

TYPES OF COSTS:

We must take into account several cost factors that weigh heavily, including fixed, off-budget, sunk, and switching costs. In a direct comparison of two new systems, where things are equal, open source software will be less expensive in almost every case. But often the comparison is in some sense a migration, where there will be a big advantage to the incumbent, most likely Windows today. This is what most TCO studies comparing proprietary software against open source actually do. In a migration, assumptions favoring the incumbent product will increase staffing costs and probably dominate the software savings. There are even some incumbent

advantages to Windows in a new installation situation, at least perceived. Decision makers may be unfamiliar with open source and inclined to assign higher risks or expect to pay more for services.

1.FixedCosts

IT and higher management costs above the level of project management are a given and will not vary based on project activities. Network and desktop infrastructure, including firewalls, storage area networks, and personal computers, can be treated as a fixed cost when looking at applications in most contemporary organizations. If we are funding a project that brings technology to a new population, we will have to consider these costs in the project, but they will in any case be the same for open source or closed code.

2.Off-BudgetCosts

End-user costs, including training, possible downtime or dissatisfaction, and self-supporting ("messing around"), can be important, and some cost models show these costs as the highest single cost component. However, they are not usually reported as costs by IT organizations, because they are not on the budget.

The effect on users may sometimes be reflected in penalties related to a service-level agreement, but more commonly as a constraint on the IT organization, which must maintain a particular level of service. The effect of off-budget costs, when included, is to make estimates of user downtime and dissatisfaction the largest elements of the cost models, although these are very difficult to measure objectively.

These elements are not included in the models here. Instead it is assumed that the systems being compared will offer equivalent availability and ease of use. This is very likely to be true for server systems, which run on the same hardware and are not directly visible to the end user. It is less easy to demonstrate for desktop systems, and may be a factor to consider. Presumably, organizations that do not find desktop systems equivalent in this regard will not deploy them regardless of cost savings.

3.SunkCosts

Sunk costs are the costs already spent on existing systems and are not recoverable. It is difficult

to get money for old systems, particularly after the dotcom bubble; many systems that are a couple of years old are only worth about ten cents on the dollar on the hardware, as a quick search on eBay will reveal. Software and support costs and other soft costs, such as training, can add up to much more than hardware and they will never be recovered. The effect of sunk costs is to make it much more difficult to move to new technology, because the acquisition cost of the new system is compared with the residual value of the existing system, which is much less than it cost.

4. Switching Costs

Switching costs are the additional costs it will take to move from an existing system to a proposed new one, as opposed to keeping the existing one. The effect of switching costs is to make new technology harder to adopt. The first application with a new technology will cost more than subsequentones, because of training of developers and administrators, who presumably know the old technology, and because of first purchase of servers, development tools, and other infrastructure that will be reused for future applications. If there were a single standard before, then adding the new technology also leads to having to support two technologies, which may lead to additional cost.

LICENSE:

Software licensing has always been a part of the process of managing systems. The issues around open source licensing are not really different from licensing in general, but they do seem to receive more attention at the moment.

Many professionals find legal issues and, in particular, licensing, one of their least favorite parts of the job. However, it is essential for all of us to know the basics of licensing. We will cover the basics in a simple way here.

If your needs are more complex, you will require a lawyer. If open source licensing documents seem long and difficult to read, you are probably just not used to reading legal documents. Typical closed code licensing agreements such as those from Microsoft or Oracle are

no better.
They are usually longer and more difficult, and very often more restrictive.

TYPES OF LICENSES

1.LICENSE

Open source licenses can be divided into two groups: the reciprocal or "free" licenses, of which the GNU General Public License (GPL) is best known, and the nonreciprocal or "open" licenses, such as the BSD and Apache licenses.

Reciprocal licenses contain a provision that requires that on relicensing the code must be open source. This is reciprocal in the sense that if a distributor receives the source code, then it passes it on to others. For example, Linux uses the GPL. If you choose to distribute an operating system based on Linux with some changes you have contributed, you must distribute the source code to that system.

Nonreciprocal licenses do not contain a relicensing provision, so they allow derivative works from open source code to revert to closed. This is nonreciprocal in the sense that a distributor can receive source code but may not necessarily pass it on. So, for example, Apple uses FreeBSD code as part of Mac OS X without needing to distribute the Mac OS X source code.

1.1 Relicensing Only Matters If You Distribute

Some people use the term *viral* for reciprocal. The implication is that handling viral licenses is dangerous, as Microsoft sometimes suggests. It is true that Microsoft needs to be careful using products licensed with the GPL.
Microsoft is a distributor of products, such as compilers and operating systems, which could appear to be derivatives. This is a risk it can handle:
Microsoft actually distributes a product (Microsoft Services for UNIX) that includes components licensed under the GPL. This risk only applies to organizations that are distributing software that extends the GPL-licensed product. Software companies that distribute code based partly on GPLlicensed products need to establish guidelines on their use.

1.2 Reciprocal Licenses Are Similar to Commercial Licenses

Reciprocal licenses are quite similar to commercial (closed source) licenses, which commonly contain terms that restrict relicensing and distribution of information. A common commercial restriction prevents you from relicensing the software or derivative works. The GPL has provisions that affect your subsequent licensing of derivative works, which is less restrictive than preventing relicensing. Commercial licenses normally require you to agree not to disclose proprietary information that you acquired under the license to others. This may include elements of source code (such as APIs) and other information such as performance data. The GPL requires that you agree to disclose the source code you acquired, and any you have added, to others.

2. LICENSES IN USE

There are many licenses in use today, but only a few that need to be considered by most organizations. The Freshmeat site lists about 50 categories of licenses, some of which are groups of licenses, but only about 20 are used by at least 100 projects. Figure 13.1 shows the distribution of licenses as reported on Freshmeat. Over two-thirds use the GPL, and about one-sixth use one of the LGPL, BSD, Apache, Mozilla, or MIT licenses. One of these five licenses should suffice for most purposes.

2.1ReciprocalLicenses

The GPL is the original "free software" license. It is used by Linux and many other core tools and will be used by everyone at some time. The GPL is also an important piece of work in its own right, and a source of controversy in some quarters, so everyone should read it. The Mozilla Public License is similar to the GPL but with clearer terms in requiring future free use.

2.2NonreciprocalLicenses

The other licenses (LGPL, BSD, Apache, and MIT) are nonreciprocal. The Lesser General Public License (LGPL) is a nonreciprocal version of the GPL intended for certain libraries. There are two forms of the BSD license. The new form omits an advertising clause in the license that was officially rescinded when the Director of the Office of Technology Licensing of the University of California stated on July 22, 1999 that clause 3 was "hereby deleted in its entirety." The new BSD license is thus equivalent to the MIT license, except for a no-

endorsement final clause. The MIT license is best known for its use in the X Windows System. The Apache license is very similar.

The nonreciprocal licenses are less restrictive than the GPL on distributors. Subsequent users can use, modify, and redistribute the code without distributing their source code. This lack of restriction for the distributor removes the rights of users downstream from them to see that code. The restriction that the GPL places on distributors has the effect of later users retaining their rights to view and modify code.
Some companies take open source software, add little or nothing, and resell the result as a closed code solution, possibly for substantial prices. Reciprocal licenses address this by ensuring that companies cannot extend code without giving it back for others to offer also. Of course, these companies may add value by improving support, documentation, bundling the product for a particular market, or developing a complementary product. They just cannot gain a proprietary advantage from changes to the code, since those enhancements must go back to the community.

2.3Which License to Use
It is strongly recommended that if you are distributing your own open source product you adopt one of these licenses without alteration:
- GNU General Public License
- Mozilla Public License
- BSD, Apache, or MIT license
- GNU Lesser General Public License

The alternative is to hire an attorney who specializes in these issues to develop a custom license, as large companies such as IBM do.

3.MIXING OPEN AND CLOSED CODE

It is quite possible to use closed code and some open source software together. This is common today and is likely to be the way most systems are built in the future.

The majority of open source developers spend most of their time on closed code development. Most open source developers work primarily on internal or closed code development within companies, so they are quite familiar with closed code.
Most open source products above the operating system are offered on one or more closed platforms, generally Windows and UNIX. Products that use databases often support some closed

code databases, most often Oracle.
Open source products are often sold as part of a bundled sale, which includes closed code products. Large organizations often purchase the top level of the software and service stack from a major closed code vendor.
Their primary purchase might be outsourcing or other services from IBM Global Services, Accenture, or CSC; software from IBM, BEA, Oracle, or SAP; hardware from IBM, HP, or Dell, with Linux and other open source products included in the overall sale.
Table shows examples of open source and closed code deployed together.
The most common hybrid case is simply organizations that obtain a variety of open source and closed code products, and then deploy them to meet *Using Open Source and Closed Code Together*

Product	Example
Compiere	An open source ERP system built on Java (closed code) and Oracle(closed code).
SAP	Closed code ERP system available on Linux and other operating systems.
SAP	converted its internal database, SAP DB, to open source and gave it to MySQL to manage (as Max DB).
Apple OS	X Closed code operating system (charging license fees) based in large part on the open source FreeBSD. Apple distributes an open source operating system called Darwin without the Apple GUI, as well as its own distribution of Xfree86.
Oracle	Closed code database (charging license fees) available on Linux (open source), as well as Windows, UNIX, and other systems.
DB2	Closed code database available on Linux, Windows, Solaris, and
IBM	operating systems.
WebSphere	IBM brand for a variety of middleware products. Includes many components, some of which are open source, such as Linux and Apache.

Their own internal needs or their customers' needs. Google, for example, employs a great deal of open source software in systems development. Its own software is not open source, and there are license restrictions on access to most Google services to prevent others from getting a free ride—for instance, by republishing a Google search as their own.

As this example shows, a company can develop closed code software using open source tools and distribute it on open source systems, as long as it follows a few simple rules. Enhancements that an organization makes to the open source software it uses, however, MUST be contributed back. It is common for companies to take open source projects, add a layer of additional functionality in closed code, offer support for both their enhancements and the open source base, and charge a fee. This describes IBM WebSphere, Red Hat, and some other distribution companies. It is a healthy part of the process, because customers have a choice of whether to choose the enhanced bundle or the open source system.

1.4 DUAL LICENSING

Some products are dual licensed. They are available with either an open source license or a commercial license. Examples of such products are:

Qt, from TrollTech, the GUI toolkit used by KDE
MySQL, from MySQL AB, the database server
Berkeley DB, from SleepyCat Software, the embedded database program

The dual license allows these companies to offer open source products to those who are developing open source software, or to individual end users. Depending on their intentions or organization, others may be required to pay for a commercial license.

There are probably many ways to do this, but the path taken by these three companies is to license under the GPL, and then offer a commercial license to companies that would prefer not to meet the GPL terms. This exact strategy requires a development tool, such as a toolkit or database; it leverages the property of the GPL so that if you link to it you fall under its terms. Vendors of a pure application might need to write different licensing terms, but of course they could.

A dual license strategy relies on code ownership. It will be difficult in practice to get a large group of contributors to assign ownership to a commercial organization. In fact, the cases of dual licensing listed, and others I know of,

have the look of a commercial company, where development is done in house and external contributions are signed over and compensated for.

UNIT III

POSSIBLE QUESTIONS

(8 MARKS)

1. Explain about open source server applications in detail.
2. Explain about "where" of OSS and " when" of OSS in detail.
3. Briefly describe about open source desktop applications.
4. Write a note on
    a. (i)Economic micro level and macro level(individual) motivation
    b. (ii) Socio political micro level and macro level(individual) motivation

5.Explain the types of costs licensing and Dual Licensing

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
COIMBATORE-641021
Department of Computer Science
III B.Sc(CS) (BATCH 2015-2018)
Open Source Software
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS **ONE MARK QUESTIONS**

UNIT III

S.No	Question	Option1	Option2	Option3	Option 4			Answer
1	_____ has a BSD-type open source license, making it alternative for both commercial and non-commercial applications	TCL	Apache	Perl	PHP			Apache
2	Apache is a _____	Web browser	Web server	Application server	Database server			Web server
3	_____ receives the request from the client	Host	Sever	Node	browser			Server
4	_____ tells the client how to interpret the information that is to follow.	Header	Footer	Content	Command			Header
5	The header is separated from the content by a _____	single white space	tab space	double white space	blank line			blank line
6	To check whether Apache is running, when starting the machine by load _____ Url in the browser	http://localhost	http://www.localhost/	http:\\localhost	http:\\www.localhost\			http://localhost
7	URL stands for _____	Uniform Resource Loader	Unified Resource Locator	Uniform Resource Locator	Unified Resource Loader			Uniform Resource Locator
8	Apache can process _____ request at a time.	one	two	Less than one	more than one			more than one
9	The Apache log file does not contain_____	Client IP address	date	the author name	the referer			the author name

10	The _____ module allows users to serve web content without having access to the main web directory.	user_mod	mod_user	mod#user	user#mod			mod_user
11	The Apache logs are located at_____	/var/log/httpd/access_log	/var/httpd/access_log	/var/access_log	/var/log/access_log			/var/log/httpd/access_log
12	_____ keeps detailed logs of accesses to the web site, errors and more.	CGI	HTML	Apache	HTTP			Apache
13	CGI stands for_____	Common Graphics Information	Common Graphics Interface	Common Gateway Interconnect	Common Gateway Interface			Common Gateway Interface
14	The locations of Apache logs are configurable in _____	httpd.config	httpd.confi	httpd.conf	httpd.con			httpd.conf
15	_____ is a log monitor program monitors the log files for security violations problems	log	watch	logwatch	watching			logwatch
16	Create a new user with _____,with a locked account to run Apache	adduser	useradd	createuser	usercreate			useradd
17	CLF stands for_____	Common Log Format	Computer Log Format	Common Logical Format	Computer Logical Format			Common Log Format
18	Which of the following is not a part of the common log format?	Requesting host	Data of request	Time to serve request	http status code			Time to serve request
19	A software that has both proprietary license and open source license is called _____.	multiple licensed software	uncontrolled software	dual licensed software	mixed licensed software			dual licensed software
20	Software with academic license _____.	does not provide the source code to the user	requires just an acknowledgement	can be used by the companies	can be used by anyone			requires just an acknowledgement

21	Dual licensing business rely on open source as _____.	production strategy	development strategy	distribution strategy	production and distribution strategy			distribution strategy
22	Open source software are distributed mainly through _____.	CDs	Internet	Memory cards.	HDDs.			Internet
23	Dual licensing works for _____.	work having many contributors	collaboratively developed softwares	single, well defined owner of a work	open source software following strategy			collaboratively developed softwares
24	_____ is a reciprocal license.	GPL	BSD License	Ms-RSL	Ms-LPL			GPL
25	Why software licensing revenue is considered as a good revenue?	Licensed software can be sold for more value	Licensing a software is very easy	Licensing a second copy of the software is possible without any additional cost	Licensed software are accepted good than others in the market			Licensing a second copy of the software is possible without any additional cost
26	Ownership applies to _____.	tangible properties only	both tangible and intangible properties	intangible properties only	software only			both tangible and intangible properties
27	Which clarifies the issues and resolves disputes among author and reader?	License	Trademark	Warranty	Copyright law			Copyright law
28	Which license enforce sharing?	reciprocal license	Academic license	Berkley Software Distribution	Proprietary license			reciprocal license
29	Which of the following is an academic license?	PUL	GPL	SPL	BSD			BSD

30	Proprietary licenses _____ for that consideration.	have some rights and pay a fee	have no rights and pay little fee	have some rights and pay no fee	have no rights and pay no fee			have some rights and pay a fee
31	Making a software available on open source terms, creates it with a _____.	small and expensive distributio n channel	large and inexpensi ve distributi on channel	large and expensiv e distributi on channel	small and inexpen sive distribut ion channel			large and inexpen sive distribut ion channel
32	Reciprocity encourages _____.	isolation	distributi on	collabora tion	Installe d base			collabor ation
33	What is warranty?	Rules	Promise	Rights	Offers			Promise
34	Which of the following provides clear warranties?	Proprietary software	Open source software.	Free software	As is' softwar e			Propriet ary software
35	Which one of the following can be patented?	Hardware	Software	Things	Names			Softwar e
36	Tomcat is a	Applicatio n server	Web Server	Desktop Server	Commu nity Server			Applicat ion server
37	Tomcat can best be described as a ?	Java Compiler	.Net Container	.Net Compiler	Servlet Contain er			Servlet Contain er
38	Apache is best described as a ?	Web Browser	Web Server	Applicati on Server	Operati ng System			Web Server
39	Linux OS is?	Proprietary OS	Open Source OS	Spread Sheet	Docum ent Viewer			Open Source OS
40	ISCII is _____?	4 bit	8 bit	16 bit	32 bit			8 bit

UNIT IV

SYLLABUS

Perl Programming
Perl - Introduction, Perl Basics: - Syntax, Variables, Strings, Numbers, Operators, Arrays: - Using Arrays, Manipulating Arrays, Associative Arrays, Chop, Length, and Sub string. Hashes, Arguments, Logic, Looping, Files, Pattern Matching, Environment Variables, Using cgilib for Forms.

PERL INTRODUCTION:

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

What is Perl?

- Perl is a stable, cross platform programming language.
- Though Perl is not officially an acronym but few people used it as **Practical Extraction and Report Language**.
- It is used for mission critical projects in the public and private sectors.
- Perl is an *Open Source* software, licensed under its *Artistic License*, or the *GNU General Public License (GPL)*.
- Perl was created by Larry Wall.
- Perl 1.0 was released to usenet's alt.comp.sources in 1987.
- At the time of writing this tutorial, the latest version of perl was 5.16.2.
- Perl is listed in the *Oxford English Dictionary*.

PC Magazine announced Perl as the finalist for its 1998 Technical Excellence Award in the Development Tool category.

Perl Features

- Perl takes the best features from other languages, such as C, awk, sed, sh, and BASIC, among others.
- Perls database integration interface DBI supports third-party databases including Oracle, Sybase, Postgres, MySQL and others.
- Perl works with HTML, XML, and other mark-up languages.
- Perl supports Unicode.
- Perl is Y2K compliant.
- Perl supports both procedural and object-oriented programming.
- Perl interfaces with external C/C++ libraries through XS or SWIG.
- Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network (CPAN).
- The Perl interpreter can be embedded into other systems.

Perl and the Web

- Perl used to be the most popular web programming language due to its text manipulation capabilities and rapid development cycle.
- Perl is widely known as "the duct-tape of the Internet".
- Perl can handle encrypted Web data, including e-commerce transactions.
- Perl can be embedded into web servers to speed up processing by as much as 2000%.
- Perl's mod_perl allows the Apache web server to embed a Perl interpreter.
- Perl's DBI package makes web-database integration easy.

Perl is Interpreted

Perl is an interpreted language, which means that your code can be run as is, without a compilation stage that creates a non portable executable program.

Traditional compilers convert programs into machine language. When you run a Perl program, it's first compiled into a byte code, which is then converted (as the program runs) into machine instructions. So it is not quite the same as shells, or Tcl, which are **strictly** interpreted without an intermediate representation.

It is also not like most versions of C or C++, which are compiled directly into a machine dependent format. It is somewhere in between, along with *Python* and *awk* and Emacs .elc files.

PERL BASICS

SYNTAX

Perl borrows syntax and concepts from many languages: awk, sed, C, Bourne Shell, Smalltalk, Lisp and even English. However, there are some definite differences between the languages

A Perl program consists of a sequence of declarations and statements, which run from the top to the bottom. Loops, subroutines, and other control structures allow you to jump around within the code. Every simple statement must end with a semicolon (;).

Perl is a free-form language: you can format and indent it however you like. Whitespace serves mostly to separate tokens, unlike languages like Python where it is an important part of the syntax, or Fortran where it is immaterial.

perl-syntax

PERL follows a very specific syntax not unlike other programming languages. It is important to develop good syntax habits as it will save you from having to debug things later, not to mention save yourself from eye strain and mind numbing headaches.

perl-casesensitivity

File names, variables, and arrays are all case sensitive. If you capitalize a variable name when you define it, you must capitalize it to call it.

A great tip for large scripts containing a vast number of variable names it is best to be consistent with your case sensitivity and maybe even develop a system for naming variables that makes sense to you. For the majority of us programmers, capitals are simply not an option.

casesensitivity.pl:

```
$VAriaBLE_NAmES = "string";
$LIKe_tHESE = "Another String";
$ARe_HArd_to_Type = "A Third String";
```

perl-comments

As with any programming language, PERL offers an escape from your code via the '#' sign. Any words, spaces, or marks after a pound symbol will be ignored by the program interpreter, offering you the coder, a chance to place reminders to yourself about your code. It's a great way to note specifics of your code to yourself or others viewing your code/script. Comments are necessary for any script you wish to publish to others or make readily available.

PERL Comment:

```
#!/usr/bin/perl

print "Content-type: text/html \n\n"; # the header
########################################
#Comments start with a #
########################################
```

This comment is extreme and overdone, you might see more comments like this in scripts that are offered free on the internet. Often programmers will include a large commented section as an installation or set-up guide included right there in the script itself.

perl-escaping characters

In PERL we use the backslash (\) character to escape any type of character that might interfere with our code. For example there may become a time when you would like to print a dollar sign rather than use one to define a variable. To do this you must "escape" the character using a backslash (\).

escapecharacters.pl:

```
#!/usr/bin/perl

print "Content-type: text/html \n\n"; #HTTP HEADER

#CREATE STRINGS WITH ESCAPING CHARACTERS
$string = "David paid \$4.34 for Larry\'s shirt.";
$email = "youremail\@youremail.com";

#PRINT THE STRINGS
print "$string<br />";
print "$email<br />";
print '$string and $email';
```

escapecharacters.pl:

David paid $4.34 for Larry's shirt.
youremail@youremail.com
$string and $email

VARIABLES:

perl-definesomevariables

A variable is defined by the ($) symbol (scalar), the (@) symbol (arrays), or the (%) symbol (hashes).

Here is what each type of variable should look like inside of a script.

definevariables.pl:

```
#!/usr/bin/perl

print "Content-type: text/html \n\n"; #HTTP HEADER

$somenumber = 4;
$myname = "some string";
@array = ("value00","value01","value02");
%hash = ("Quarter", 25, "Dime", 10, "Nickle", 5);
## OR ##
```

```
my $somenumber = 4;
my $myname = "some string";
my @array = ("value00", "value01", "value02");
my %hash = ("Quarter", 25, "Dime", 10, "Nickle", 5);
```

The latter example using the *my* parameter is another means to define a variable that you might run across as you gain more experience. It is not necessary to use the *my* parameter. Variables can be defined either way.

perl-scalarvariables

Scalar variables are simple variables containing only one element--a string, a number, or a reference. Strings may contain any symbol, letter, or number. Numbers may contain exponents, integers, or decimal values. The bottom line here with scalar variables is that they contain only one single piece of data. What you see is what you get with scalar variables.

definescalars.pl:

```
#!/usr/bin/perl

print "Content-type: text/html \n\n"; #HTTP HEADER

# DEFINE SOME SCALAR VARIABLES
$number = 5;
$exponent = "2 ** 8";
$string = "Hello, Perl!";
$stringpart_1 = "Hello, ";
$stringpart_2 = "Perl!";
$linebreak = "<br />"; #HTML LINEBREAK TAG

# PRINT THEM TO THE BROWSER
print $number;
print $linebreak;
print $exponent;
print $linebreak;
print $string.$linebreak;
print $stringpart_1.$stringpart_2;
```

Display:

5
2 ** 8
Hello, Perl!
Hello, Perl!

Scalars are very straight forward. Notice that we used a period (.) between each of our variables. This is a special kind of *operator* that concatenates two or more variables.

perl-arrayvariables

Arrays contain a list of scalar data (single elements). A list can hold an unlimited number of elements. In Perl, arrays are defined with the at (@) symbol.

definearrays.pl:

```
#!/usr/bin/perl

print "Content-type: text/html \n\n"; #HTTP HEADER

#DEFINE SOME ARRAYS
@days = ("Monday", "Tuesday", "Wednesday");
@months = ("April", "May", "June");

#PRINT MY ARRAYS TO THE BROWSER
print @days;
print "<br />";
print @months;
```

Display:

MondayTuesdayWednesday
AprilMayJune

perl-definea hash

Hashes are complex lists with both a *key* and a *value* part for each element of the list. We define a hash using the percent symbol (%).

definehashes.pl:

```
print "Content-type: text/html \n\n"; #HTTP HEADER

#DEFINE SOME HASHES
%coins = ("Quarter", 25, "Dime", 10, "Nickle", 5);
%ages = ("Jerry", 45, "Tom", 22, "Vickie", 38);

#PRINT MY HASHES TO THE BROWSER
print %coins;
print "<br />";
print %ages;
```

Display:

Dime10Nickle5Quarter25
Jerry45Vickie38Tom22

Hashes are very complex data types, for now just understand the syntax of how to define one. Later we will take a closer look at these complex variables.

STRINGS:

perl-strings

Strings are scalar as we mentioned previously. There is no limit to the size of the string, any amount of characters, symbols, or words can make up your strings.

When defining a string you may use single or double quotations, you may also define them with the *q* subfunction.

definestrings.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #HTTP HEADER

# DEFINE SOME STRINGS
$single = 'This string is single quoted';
$double = "This string is double quoted";
$userdefined = q^Carrot is now our quote^;
```

```
# PRINT THEM TO THE BROWSER
print $single."<br />";
print $double."<br />";
print $userdefined."<br />";
```

perl-formatting strings w/ formatting characters

Strings can be formatted to your liking using formatting characters. Some of these characters also work to format files created in PERL. Think of these characters as miniature functions.

Character	Description
\L	Transform all letters to lowercase
\l	Transform the next letter to lowercase
\U	Transform all letters to uppercase
\u	Transform the next letter to uppercase
\n	Begin on a new line
\r	Applys a carriage return
\t	Applys a tab to the string
\f	Applys a formfedd to the string
\b	Backspace
\a	Bell
\e	Escapes the next character

\0nn	Creates Octal formatted numbers
\xnn	Creates Hexideciamal formatted numbers
\cX	Control characters, x may be any character
\Q	Do not match the pattern
\E	Ends \U, \L, or \Q functions

formattingcharacters.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #HTTP HEADER

# STRINGS TO BE FORMATTED
$mystring = "welcome to tizag.com!"; #String to be formatted
$newline = "welcome to \ntizag.com!";
$capital = "\uwelcome to tizag.com!";
$ALLCAPS = "\Uwelcome to tizag.com!";

# PRINT THE NEWLY FORMATTED STRINGS
print $mystring."<br />";
print $newline."<br />";
print $capital."<br />";
print $ALLCAPS;
```

Any combination of these special characters can be used at any time to properly punctuate your strings. They also come in handy when printing out HTML with your PERL functions.

perl-substr() and string indexing

The substr() function is a rather complicated function. It can be used to do many things and we'll start with the most basic, grabbing a substring and move onto more advanced ideas further on.

To use substr() to grab a substring, you need to give it both a string variable to pick something out of and an offset (which starts at 0). A string can be thought of as an array of characters, starting with element 0 at the beginning and +1 for each additional character. The string "hey" has 3 characters. The 0th element is "h", the 1st element is "e" and the 2nd and last element is "y".

The first argument of substr() is the string we want to take something from and the second argument is the offset, or where we want to start at.

stringreplace.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #HTTP HEADER

# DEFINE A STRING TO REPLACE
$mystring = "Hello, am I about to be manipulated?!";

# PRINT THE ORIGINAL STRING
print "Original String: $mystring<br />";

# STORE A SUB STRING OF $mystring, OFFSET OF 7
$substringoffset = substr($mystring, 7);
print "Offset of 7: $substringoffset<br />";
```

Display:

Original String: Hello, am I about to be manipulated?!
Offset of 7: am I about to be manipulated?!

substr() started at the 7th element (remember we count from 0) which was the "a" in "am" and returned the rest of the string and we stored it into $substringoffset. Play around with this function a little and get a feel for how offset works!

Below we have gone on to the more advanced options of substr(), taking advantage of the last two arguments of the function: length and replace value. Rather than grabbing the whole string from the offset, we can just grab a chunk of it by specifying the length we want this substring to be.

The final argument, replace value, replaces the substring specified by the first three arguments with whatever we want. Let's change the original string to say something different by grabbing a part of the string and replacing it with "I want".

stringreplace.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #HTTP HEADER
# DEFINE A STRING TO REPLACE
$mystring = "Hello, am I about to be manipulated?!";

# PRINT THE ORIGINAL STRING
print "Original String: $mystring<br />";

# STORE A SUB STRING OF $mystring, OFFSET OF 7 AND LENGTH 10
$suboffsetANDlength = substr($mystring, 7, 10);
print "Offset of 7 and length of 10: $suboffsetANDlength<br />";

# CHANGE $mystring, OFFSET OF 7 AND LENGTH 10 AND
# REPLACE SUB STR WITH "I want"
$suboffsetANDlength = substr($mystring, 7, 10, "I want");
print "mystring is now: $mystring<br />";
```

Display:

Original String: Hello, am I about to be manipulated?!
Offset of 7 and length of 10: am I about
mystring is now: Hello, I want to be manipulated?!

The original string was changed, so be careful when using the replace value argument of substr(). However, it's a great tool to have in your arsenal, as changing strings in this manner is pretty common. Please play around with substr() for a while and make sure you understand it!

NUMBERS:

perl-numbers

Numbers are scalar data. They exist in PERL as real numbers, float, integers, exponents, octal, and hexidecimal numbers.

perlnumbers.pl:

```
$real = 27;
$float = 3.14159;
$integer = -4;
$exponent = 10e12;
```

perl-mathematicalfunctions

With numbers comes math. Simple arithmetic operations are discussed in the PERL Operators lesson.

Some mathematical functions require some additional PERL Modules. Here's a few trigonomic functions that will only function if your build of PERL has the *Math::Trig* module installed.

perltrig.pl:

```
#!/usr/bin/perl
use Math::Trig; #USE THIS MODULE

print "content-type: text/html \n\n"; #HTTP HEADER
$real = 27;
$float = 3.14159;
$integer = -4;
$exponent = 10e12;
print tan($real);     #TANGENT FUNCTION
print "<br />";
print sin($float);    #SINE FUNCTION
print "<br />";
print acos($integer); #COSINE FUNCTION
```

perltrig.pl:

```
-3.27370380042812
2.65358979335273e-06
3.14159265358979-2.06343706889556i
```

perl-numbers with operators

Numbers aren't much without arithmetic operations. This next example is a sneak peak of the next lesson, PERL Operators.

arithmeticoperations.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #HTTP HEADER

#PICK TWO NUMBERS
$x = 14;
$y = 10;
#MULTIPLICATION OPERATOR
$area = ($x * $y);
print $area;
print "<br />";
```

arithmeticoperations.pl:

140

perl-formatting numbers

Computers are capable of calculating numbers that you and I probably never knew existed. This is especially true with calculations involving decimals, floating-point numbers, or percentages.

You may find that one of the best solutions is to first convert your numbers when possible to integers (get rid of the decimal). You may then go ahead and perform the required operations such as multiplication, division, addition, or whatever and finally reintroduce the decimal using division.

peskydecimals.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #HTTP HEADER

$hourlyrate = 7.50; #DECIMAL TO BE RID OF
```

```
$hoursworked = 35;
$no_decimal_rate = ($hourlyrate * 100);

$netpay = ($no_decimal_rate * $hoursworked);
$paycheck = ($netpay / 100);

print "Hourly Wage: $hourlyrate<br />";
print "Hours: $hoursworked<br />";
print "No Decimal: $no_decimal_rate<br />";
print "Net Pay: $netpay<br />";
print "Pay Check: $paycheck<br />";
```

peskydecimals.pl:

Hourly Wage: 7.5 Hours: 35

No Decimal: 750

Net Pay: 26250

Pay Check: 262.5

In this example we followed the steps stated above, first we removed the decimal from each number involved in the calculation, ($hourlyrate and $hoursworked). Then we performed the operation ($netpay), and finally introduced the decimal again by dividing our $netpay by the same number we used to get rid of the decimal in the first place (100).

- Convert to real numbers.
- Perform the operation(s).
- Convert back to a decimal.

NUMBERS:

perl-arithmetic operators

Arithmetic operators are symbols used to execute general arithmetic procedures including: addition (+), subtraction (-), multiplication (*), and division (/).

Arithmetic Operators:

Operator	Example	Result	Definition
+	7 + 7	= 14	Addition
-	7 - 7	= 0	Subtraction
*	7 * 7	= 49	Multiplication
/	7 / 7	= 1	Division
**	7 ** 7	= 823543	Exponents
%	7 % 7	= 0	Modulus

With these operators we can take a number and perform some simple math operations.

PERL Arithmetic:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";    #HTTP Header

#PICK A NUMBER
$x = 81;
$add = $x + 9;
$sub = $x - 9;
$mul = $x * 10;
$div = $x / 9;
$exp = $x ** 5;
$mod = $x % 79;
print "$x plus 9 is $add<br />";
print "$x minus 9 is $sub<br />";
print "$x times 10 is $mul<br />";
print "$x divided by 9 is $div<br />";
print "$x to the 5th is $exp<br />";
print "$x modulus 79 is $mod<br />";
```

Your browser should read:

arithmetic.pl:
81 plus 9 is 90
81 minus 9 is 72
81 times 10 is 810
81 divided by 9 is 9
81 to the 5th is 3486784401
81 modulus 79 is 2

perl-assignment operators

Assignment operators perform an arithmetic operation and then assign the value to the existing variable. In this example, we set a variable ($x) equal to 5. Using assignment operators we will replace that value with a new number after performing some type of mathematical operation.

Assignment Operators:

Operator	Definition	Example
+=	Addition	($x += 10)
-=	Subtraction	($x -= 10)
*=	Multiplication	($x *= 10)
/=	Division	($x /= 10)
%=	Modulus	($x %= 10)
**=	Exponent	($x **= 10)

PERL Assignment:

```
#!/usr/bin/perl
```

```
print "content-type: text/html \n\n";    #HTTP HEADER

#START WITH A NUMBER
$x = 5;
print '$x plus 10 is '.($x += 10);
print "<br />x is now ".$x;       #ADD 10
print '<br />$x minus 3 is '.($x -= 3);
print "<br />x is now ".$x;       #SUBTRACT 3
print '<br />$x times 10 is '.($x *= 10);
print "<br />x is now ".$x.       #MULTIPLY BY 10
print '<br />$x divided by 10 is '.($x /= 10);
print "<br />x is now ".$x;       #DIVIDE BY 10
print '<br />Modulus of $x mod 10 is '.($x %= 10);
print "<br />x is now ".$x;       #MODULUS
print '<br />$x to the tenth power is '.($x **= 10);
print "<br />x is now ".$x;       #2 to the 10th
```

Display:

$x plus 10 is 15
x is now 15
$x minus 3 is 12
x is now 12
$x times 10 is 120
$x is now 120
$x divided by 10 is 12
x is now 12
Modulus of $x mod 10 is 2
x is now 2
$x to the tenth power is 1024
x is now 1024

Each time an operation is performed our variable ($x) is permanently changed to a new value of $x.

perl-logical & relational operators

Relationship operators compare one variable to another. (5 < 12) They are used to compare equality or inequality of two or more variables, be it a string or numeric data.

Logical operators state *and/or* relationships. Meaning, you can take two variables and test an either or conditional. Logical operators are used later on in conditionals and loops. For now, just be able to recognize them in the upcoming examples.

Logical/Relational Operators:

Relational

Operator	Example	Defined	Result
==,eq	5 == 5 5 eq 5	Test: Is 5 equal to 5?	True
!=,ne	7 != 2 7 ne 2	Test: Is 7 not equal to 2?	True
<,lt	7 < 4 7 lt 4	Test: Is 7 less than 4?	False
>,gt	7 > 4 7 gt 4	Test: Is 7 greater than 4?	True
<=,le	7 <= 11 7 le 11	Test: Is 7 less than or equal to 11?	True
>=,ge	7 >= 11 7 ge 11	Test: Is 7 greater than or equal to 11?	False

Logical

Operator	Defined	Example
&&,and	Associates two variables using AND	if (($x && $y) == 5)...
||,or	Associates two variables using OR	if (($x || $y) == 5)...

Please note that you must use each different operator depending of whether or not you are comparing strings or numbers. In the table above, the black operators are for numbers and the red ones are for strings.

perl-variables+operators

Variables can be used with mathematical formulas using PERL Operatorsdiscussed in a previous lesson. Also, note that variables are case sensitive. "$myvariable," "$MYvariable," and "$Myvariable" can all be assigned different values due to case sensitivity. Numbers of course can be added, subtracted, or multiplied using operators. Strings as shown in the example below can also be used with operators.

PERL Code:

```
#!/usr/bin/perl

print "Content-type: text/html \n\n"; #HTTP HEADER

#TWO STRINGS TO BE ADDED
$myvariable = "Hello,";
$Myvariable = " World";

#ADD TWO STRINGS TOGETHER
$string3 = "$myvariable $Myvariable";

print $string3;
```

Display:
Hello, World

ARRAYS:

perl-arrayvariables

Arrays are a special type of variable that store list style data types. Each object of the list is termed an element and elements can either be a string, a number, or any type of scalar data including another variable.

Place an array into a PERL script, using the *at* symbol (@).

perlarrays.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";    #HTTP HEADER

# DEFINE AN ARRAY
@coins = ("Quarter","Dime","Nickel");

# PRINT THE ARRAY
print "@coins";
print "<br />";
print @coins;
```

Check the syntax here. We printed the same array twice using quotes around the first line. Notice how the line with quotes around it prints nicely to the browser leaving spaces between each word. PERL does this automatically as it assumes the quotations are meant for a string and strings are usually comprised of words that require spacing between each word.

perl-arrayindexing

Each element of the array can be indexed using a scalar version of the same array. When an array is defined, PERL automatically numbers each element in the array beginning with zero. This phenomenon is termed *array indexing*.

arrayindexing.pl:

```
#!/usr/bin/perl
```

```
print "content-type: text/html \n\n";    #HTTP HEADER

# DEFINE AN ARRAY
@coins = ("Quarter","Dime","Nickel");

# PRINT THE WHOLE ARRAY
print "@coins";

# PRINT EACH SCALAR ELEMENT
print "<br />";
print $coins[0]; #Prints the first element
print "<br />";
print $coins[1]; #Prints the 2nd element
print "<br />";
print $coins[2]; #Prints the 3rd element
```

arrayindexing.pl:

Quarter Dime Nickel
Quarter
Dime
Nickel

Elements can also be indexed backwards using negative integers instead of positive numbers.

MANIPULATING ARRAYS:

pop

The pop function will remove and return the last element of an array.

In this first example you can see how, given an array of 3 elements, the pop function removes the last element (the one with the highest index) and returns it.

```
1.  my @names = ('Foo', 'Bar', 'Baz');
2.  my $last_one = pop @names;
3.
4.  print "$last_one\n";  # Baz
5.  print "@names\n";     # Foo Bar
```

In the special case of the original array being empty, the pop function will return undef.

push

The push function can add one or more values to the end of an array. (Well, it can also add 0 values, but that's not very useful, is it?)

```
1.  my @names = ('Foo', 'Bar');
2.  push @names, 'Moo';
3.  print "@names\n";     # Foo Bar Moo
4.
5.  my @others = ('Darth', 'Vader');
6.  push @names, @others;
7.  print "@names\n";     # Foo Bar Moo Darth Vader
```

n this example we originally had an array with two elements. Then we pushed a single scalar value to the end and our array got extended to a 3-element array.

In the second call to push, we pushed the content of the @others array to the end of the @names array, extending it to a 5-element array.

shift

If you imagine the array starting on the left hand side, the shift function will move the whole array one unit to the left. The first element will "fall off" the array and become the function's return value. (If the array was empty, **shift** will return undef.)

After the operation, the array will be one element shorter.

```
1.  my @names = ('Foo', 'Bar', 'Moo');
2.  my $first = shift @names;
3.  print "$first\n";     # Foo
4.  print "@names\n";     # Bar Moo
```

This is quite similar to pop, but it works on the lower end of the array.

unshift

This is the opposite operation of shift. unshift will take one or more values (or even 0 if that's what you like) and place it at the beginning of the array, moving all the other elements to the right.

You can pass it a single scalar value, which will become the first element of the array. Or, as in the second example, you can pass a second array and then the elements of this second array (@others in our case) will be copied to the beginning of the main array (@names in our case) moving the other elements to higher indexes.

```
1.  my @names = ('Foo', 'Bar');
2.  unshift @names, 'Moo';
3.  print "@names\n";     # Moo Foo Bar
4.
5.  my @others = ('Darth', 'Vader');
6.  unshift @names, @others;
7.  print "@names\n";     # Darth Vader Moo Foo Bar
```

ASSOCIATIVE ARRAYS

Associative arrays are a very useful and commonly used feature of Perl.

Associative arrays basically store ***tables*** of information where the lookup is the right hand ***key*** (usually a string) to an associated scalar value. Again scalar values can be mixed ``types".

We have already been using Associative arrays for name/value pair input to CGI scripts.

Associative arrays are denoted by a verb| When you declare an associative array the key and associated values are listed in consecutive pairs.

So if we had the following secret code lookup:

name	code
dave	1234

peter	3456
andrew	6789

We would declare a Perl associative array to perform this lookup as follows:

```
%lookup = ("dave", 1234,
        "peter", 3456,
        "andrew", 6789);
```

The reference a particular value you do:

```
$lookup{"dave"}
```

You can create new elements by assignments to new ***keys***. ***E.g.***

```
$lookup{"adam"} = 3845;
```

You do new assignments to old keys also:

```
# change dave's code
$lookup{"dave"} = 7634;
```

Array Functions:

Function	**Definition**
push(@array, *Element*)	Adds to the end of an array
pop(@array)	Removes the last element of the array
unshift(@array, *Element*)	Adds to the beginning of an array
shift(@array)	Removes the first element of an array
delete $array[index]	Removes an element by index number

It is also possible to remove any element by its indexed number. Just remember to use the scalar form of the array when doing so.($)

CHOP:

Description

This function removes the last character from EXPR, each element of LIST, or $_ if no value is specified.

Syntax

Following is the simple syntax for this function −

```
chop VARIABLE

chop( LIST )

chop
```

Return Value

This function returns the character removed from EXPR and in list context, the character is removed from the last element of LIST.

Example

Following is the example code showing its basic usage −

Live Demo

```
#!/usr/bin/perl


$string1 = "This is test";

$retval  = chop( $string1 );


print " Choped String is : $string1\n";
```

```
print " Character removed : $retval\n";
```

When above code is executed, it produces the following result −

```
Choped String is : This is tes
Number of characters removed : t
```

LENGTH:

Description

This function returns the length, in characters, of the value of EXPR, or $_ if not specified. Use scalar context on an array or hash if you want to determine the corresponding size.

Syntax

Following is the simple syntax for this function −

```
length EXPR

length
```

Return Value

This function returns the size of string.

Example

Following is the example code showing its basic usage −

Live Demo

```
#!/usr/bin/perl


$orig_string = "This is Test and CAPITAL";

$string_len =  length( orig_string );
```

```
print "Length of  String is : $string_len\n";
```

When above code is executed, it produces the following result −

```
Length of  String is : 11
```

SUBSTRING:

Description

This function returns a substring of EXPR, starting at OFFSET within the string. If OFFSET is negative, starts that many characters from the end of the string. If LEN is specified, returns that number of bytes, or all bytes up until end-of-string if not specified. If LEN is negative, leaves that many characters off the end of the string.

If REPLACEMENT is specified, replaces the substring with the REPLACEMENT string.

If you specify a substring that passes beyond the end of the string, it returns only the valid element of the original string.

Syntax

Following is the simple syntax for this function −

```
substr EXPR, OFFSET, LEN, REPLACEMENT

substr EXPR, OFFSET, LEN

substr EXPR, OFFSET
```

Return Value

This function returns string.

Example

Following is the example code showing its basic usage −

```
#!/usr/bin/perl -w
```

```
$temp = substr("okay", 2);

print "Substring valuye is $temp\n";


$temp = substr("okay", 1,2);

print "Substring valuye is $temp\n";
```

When above code is executed, it produces the following result −

```
Substring valuye is ay
Substring valuye is ka
```

HASHES:

perl-hashes

Hashes are complex list data, like arrays except they link a key to a value. To define a hash, we use the percent (%) symbol before the name.

defineahash.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";

# DEFINE A HASH
%coins = ("Quarter", 25, "Dime", 10, "Nickel", 5);

# PRINT THE HASH
print %coins;
```

Display:
Nickel5Dime10Quarter25

perl-hashindexing

Hashes can be indexed using two scalar variables. The variables *$key* and *$value* can be used to call on each *key* or *value* of the hash. We can use these variables to print out our hash again but this time let's make it a little more legible using a *while* loop.

legiblehash.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";

# DEFINE A HASH
%coins = ( "Quarter" , 25,
          "Dime" ,    10,
          "Nickel",    5 );

# LOOP THROUGH IT
while (($key, $value) = each(%coins)){
    print $key.", ".$value."<br />";
}
```

legiblehash.pl:

```
Nickel,                                    5
Dime,                                      10
Quarter, 25
```

The **each()** function takes a hash. It then removes the topmost "Key and Value" pair. We store this into the variables $key and $value. Each time this loop iterates, the statement ($key, $value) = each(%thing) executes and the hash pops off the top key value pair and will continue doing so until it has gone through every pair in the hash. When it is done, *each()* returns false and the loop stops running.

Hashes work really well with HTML Tables.

tablehashes.pl:

```
#!/usr/bin/perl
```

```
print "content-type: text/html \n\n";

# DEFINE A HASH
%coins = ( "Quarter" , 25,
          "Dime" ,    10,
          "Nickel",    5 );

# SET UP THE TABLE
print "<table border='1'>";
print "<th>Keys</th><th>Values</th>";

# EXECUTE THE WHILE LOOP
while (($key, $value) = each(%coins)){
    print "<tr><td>".$key."</td>";
    print "<td>".$value."</td></tr>";
}
print "</table>";
```

tablehashes.pl:

Keys	Values
Nickel	5
Dime	10
Quarter	25

We have yet to sort our hash so you may experience different arrangements of your keys than shown in the display box.

ARGUMENTS:

Perl command line arguments stored in the special array called @ARGV. The array @ARGV contains the command-line arguments intended for the script. $#ARGV is generally the number of arguments minus one, because $ARGV[0] is the first argument, not the

program's command name itself. Please note that $ARGV contains the name of the current file when reading from <>.

To read arguments from a PERL script you use the environment variable ARGV.

Arguments are stored in the array @ARGV

First Argument: $ARGV[0]

Second Argument: $ARGV[1]

Third Argument: $ARGV[2]

etc.

Example

./script.pl arg1 num2 bob

$ARGV[0] == "arg1"

$ARGV[1] == "num2"

$ARGV[2] == "bob"

To get the total number of arguments

$numberOfArguments = $#ARGV + 1;

To read all arguments as one string

$arg_string=join(' ',@ARGV);

LOGIC

There are following logical operators supported by Perl language. Assume variable $a holds true and variable $b holds false then −

Sr.No.	Operator & Description
1	**and** Called Logical AND operator. If both the operands are true then then condition becomes true. **Example** − ($a and $b) is false.
2	**&&** C-style Logical AND operator copies a bit to the result if it exists in both operands. **Example** − ($a && $b) is false.
3	**or** Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true. **Example** − ($a or $b) is true.
4	**||** C-style Logical OR operator copies a bit if it exists in eather operand. **Example** − ($a || $b) is true.
5	**not** Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

	Example − not($a and $b) is true.

Example

Try the following example to understand all the logical operators available in Perl. Copy and paste the following Perl program in test.pl file and execute this program.

```
#!/usr/local/bin/perl

$a = true;
$b = false;

print "Value of \$a = $a and value of \$b = $b\n";

$c = ($a and $b);
print "Value of \$a and \$b = $c\n";

$c = ($a  && $b);
print "Value of \$a && \$b = $c\n";

$c = ($a or $b);
print "Value of \$a or \$b = $c\n";

$c = ($a || $b);
```

```
print "Value of \$a || \$b = $c\n";


$a = 0;

$c = not($a);

print "Value of not(\$a)= $c\n";
```

When the above code is executed, it produces the following result −

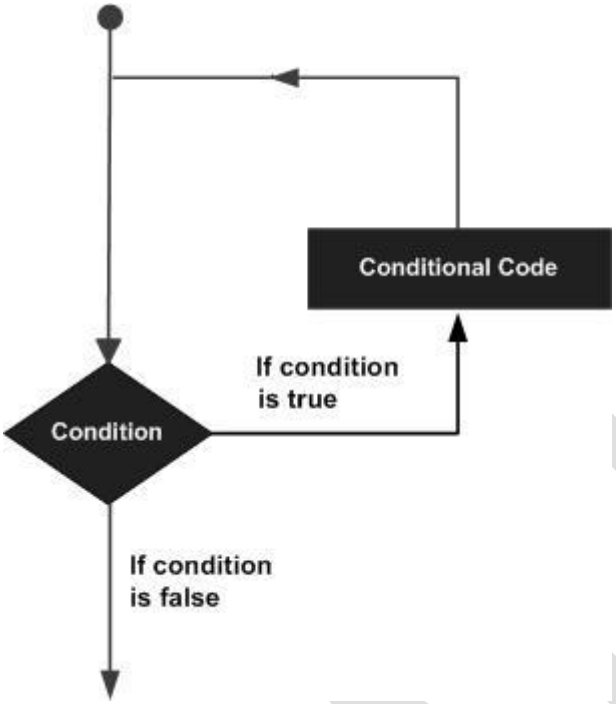
```
Value of $a = true and value of $b = false
Value of $a and $b = false
Value of $a && $b = false
Value of $a or $b = true
Value of $a || $b = true
Value of not($a)= 1
```

LOOP

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



Perl programming language provides the following types of loop to handle the looping requirements.

Sr.No.	Loop Type & Description
1	**while loop** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	**until loop** Repeats a statement or group of statements until a given condition becomes true. It tests the condition before executing the loop body.
3	**for loop** Executes a sequence of statements multiple times and abbreviates the code that

	manages the loop variable.
4	**foreach loop** The foreach loop iterates over a normal list value and sets the variable VAR to be each element of the list in turn.
5	**do...while loop** Like a while statement, except that it tests the condition at the end of the loop body
6	**nested loops** You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements

Loop control statements change the execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements. Click the following links to check their detail.

Sr.No.	Control Statement & Description
1	**next statement** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
2	**last statement** Terminates the loop statement and transfers execution to the statement immediately following the loop.
3	**continue statement**

	A continue BLOCK, it is always executed just before the conditional is about to be evaluated again.
4	**redo statement** The redo command restarts the loop block without evaluating the conditional again. The continue block, if any, is not executed.
5	**goto statement** Perl supports a goto command with three forms: goto label, goto expr, and goto &name.

The Infinite Loop

A loop becomes infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the **for** loop are required, you can make an endless loop by leaving the conditional expression empty.

```
#!/usr/local/bin/perl

for( ; ; ) {
   printf "This loop will run forever.\n";
}
```

You can terminate the above infinite loop by pressing the Ctrl + C keys.

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but as a programmer more commonly use the for (;;) construct to signify an infinite loop.

perl-forloops

A for loop counts through a range of numbers, running a block of code each time it iterates through the loop. The syntax is *for($start_num, Range, $increment) { code to execute }*. A for loop needs 3 items placed inside of the conditional statement to be successful. First a starting point, then a range operator, and finally the incrementing value. Below is the example.

forloop.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";

# SET UP THE HTML TABLE
print "<table border='1'>";

# START THE LOOP, $i is the most common counter name for a loop!
for($i = 1; $i < 5; $i++) {
        # PRINT A NEW ROW EACH TIME THROUGH W/ INCREMENT
        print "<tr><td>$i</td><td>This is row $i</td></tr>";
}
# FINISH THE TABLE
print "</table>";
```

forloop.pl:

1	This is row 1
2	This is row 2
3	This is row 3
4	This is row 4
5	This is row 5

We looped through one variable and incremented it. Using HTML, we were able to make a nice table to demonstrate our results.

perl-foreachloops

Foreach is designed to work with arrays. Say you want to execute some code *foreach* element within an array. Here's how you might go about it.

foreachloop.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";    #The header

# SET UP THE HTML TABLE
print "<table border='1'>";

# CREATE AN ARRAY
@names = qw(Steve Bill Connor Bradley);

# SET A COUNT VARIABLE
$count = 1;

# BEGIN THE LOOP
foreach $names(@names) {
        print "<tr><td>$count</td><td>$names</td></tr>";
        $count++;
}
print "</table>";
```

foreachloop.pl:

1	Steve
2	Bill
3	Connor
4	Bradley

We placed a table row counter for you to see each line more clearly. We use the variable $names to pull single elements from our array, PERL does the rest for us by looping through each element in our array. Use the sorting functions outlined in the PERL Arrays lesson.

perl-while

While loops continually iterate as long as the conditional statement remains true. It is very easy to write a conditional statement that will run forever especially at the beginner level of coding. On a more positive note, *while* loops are probably the easiest to understand. The syntax is while (conditional statement) { execute code; }.

whilecounter.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";

# SET A VARIABLE
$count = 0;

# RUN A WHILE LOOP
while ($count <= 7) {
        # PRINT THE VARIABLE AND AN HTML LINE BREAK
        print "$count<br />";
        # INCREMENT THE VARIABLE EACH TIME
        $count ++;
}
print "Finished Counting!";
```

whilecounter.pl:

```
0
1
2
3
4
5
6
```

7
Finished Counting!

perl-next,last,andredo

Outlined below are several interrupts that can be used to redo or even skip iterations of code. These functions allow you to control the flow of your *while* loops.

Next

Place it inside your loop and it will stop the current iteration and go on to the next one.

Continue

Executed after each loop iteration and before the conditional statement is evaluated. A good place to increment counters.

Last

Last stops the looping immediately (like break)

Redo

Redo will execute the same iteration over again.

flowcontrol.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";

# SET A VARIABLE
$count = 0;
while ($count <= 7) {

        # SET A CONDITIONAL STATEMENT TO INTERRUPT @ 4
        if ($count == 4) {
                print "Skip Four!<br />";
                next;
        }
        # PRINT THE COUNTER
```

```
        print $count."<br />";

}
        continue {
                $count++;
        };
print "Loop Finished!";
```

flowcontrol.pl:

```
0
1
2
3
Skip                                                    Four!
5
6
7
Finished Counting!
```

Above, we skip the fourth iteration by incrementing the variable again. In the example we also print a line, "Skip Four!" just to make things easier to follow.

perl-whilearrayloop

Here we are just showing a method of looping through an array using a while loop. We use three variables to do this including: the array, a counter, and an index number so that each time the while loop iterates we also loop through each index of the array.

whilearrayloop.pl:

```
#!/usr/bin/perl

print "content-type: text/html \n\n";

# SET UP AN HTML TABLE
print "<table border='1'>";

# DEFINE AN ARRAY
```

```
@names = qw(Steve Bill Connor Bradley);

# COUNTER - COUNTS EACH ROW
$count = 1;

# COUNTS EACH ELEMENT OF THE ARRAY
$n = 0;

# USE THE SCALAR FORM OF ARRAY
while ($names[$n]) {
        print "<tr><td>$count</td><td>$names[$n]</td></tr>";
        $n++;
        $count++;
}
print "</table>";
```

while.pl:

1	Steve
2	Bill
3	Connor
4	Bradley

FILES

The basics of handling files are simple: you associate a **filehandle** with an external entity (usually a file) and then use a variety of operators and functions within Perl to read and update the data stored within the data stream associated with the filehandle.

A filehandle is a named internal Perl structure that associates a physical file with a name. All filehandles are capable of read/write access, so you can read from and update any file or device

associated with a filehandle. However, when you associate a filehandle, you can specify the mode in which the filehandle is opened.

Three basic file handles are - **STDIN**, **STDOUT**, and **STDERR,** which represent standard input, standard output and standard error devices respectively.

Opening and Closing Files

There are following two functions with multiple forms, which can be used to open any new or existing file in Perl.

```
open FILEHANDLE, EXPR

open FILEHANDLE


sysopen FILEHANDLE, FILENAME, MODE, PERMS

sysopen FILEHANDLE, FILENAME, MODE
```

Here FILEHANDLE is the file handle returned by the **open** function and EXPR is the expression having file name and mode of opening the file.

PATTERN MATCHING:

Patterns

Patterns are subject to an additional level of interpretation as a regular expression. This is done as a second pass, after variables are interpolated, so that regular expressions may be incorporated into the pattern from the variables. If this is not what you want, use \Q to interpolate a variable literally.

?PATTERN?

This is just like the /pattern/ search, except that it matches only once between calls to the reset() operator. This is a useful optimization when you only want to see the first occurrence of something in each file of a set of files, for instance. Only ?? patterns local to the current package

are reset.

m/PATTERN/gimosx

Searches a string for a pattern match, and in a scalar context returns true (1) or false ('). If no string is specified via the =~ or !~ operator, the $_ string is searched. The string specified with =~ can be a variable or the result of an expression evaluation. The initial 'm' can be omitted if '/' is used for the delimiters, otherwise any non-alphanumeric character can be used (apart from whitespace).

The modifier options are:

- g Match globally, i.e. find all occurrences.
- i Do case-insensitive pattern matching.
- m Treat string as multiple lines - default is to assume just a single line in the string (no embedded newlines). See $*.
- o Only compile pattern once, even if variables within it change.
- s Treat string as single line.
- x Use extended regular expressions. Whitespace that is not backslashed or within a haracter class is ignored, allowing the regular expression to be broken into more readable parts with embedded comments.

In a list context, the pattern match returns the portions of the target string that match the expressions within the pattern in brackets. In a scalar context, each iteration identifies the next match (pos() holding the position of the previous match on the variable).

q/STRING/, 'STRING'

A single-quoted, literal string, default delimiters are single quotes ('...'). Backslashes are ignored, unless followed by the delimiter or another backslash, in which case the delimiter or backslash is interpolated.

qq/STRING/, "STRING"

A double-quoted, interpolated string, default delimiters are double quotes ("...").

qx/STRING/, `STRING`

A string which is interpolated and then executed as a system command. The collected standard output of the command is returned. In scalar context, it comes back as a single (potentially multi-line) string. In list context, returns a list of lines (depending on how the $/ delimiter is specified).

```
$today = qx{ date };
```

qw/STRING/

Returns a list of the words extracted out of STRING, using embedded whitespace as the word delimiters. It is exactly equivalent to:

```
split(' ', q/STRING/);
```

Some frequently seen examples:

```
use POSIX qw( setlocale localeconv )
@EXPORT = qw( foo bar baz );
```

s/PATTERN/REPLACEMENT/egimosx

Searches a string for a pattern, and if found, replaces that pattern with the replacement text and returns the number of substitutions made. Otherwise it returns false (0).

If no string is specified via the =~ or !~ operator, the $_ variable is searched and modified. (The string specified with =~ must be a scalar variable, an array element, a hash element, or an assignment to one of those, i.e. an lvalue.)

If the delimiter chosen is single quote, no variable interpolation is done on either the PATTERN or the REPLACEMENT. Otherwise, if the PATTERN contains a $ that looks like a variable rather than an end-of-string test, the variable will be interpolated into the pattern at run-time. If you only want the pattern compiled once the first time the variable is interpolated, use the /o option. If the pattern evaluates to a null string, the most recently executed (and successfully

compiled) regular expression is used instead.

The modifier options are (see m/PATTERN/ above for more detailed descriptions of common modifiers):

e Evaluate the right side as an expression.

g Match globally, i.e. all occurrences.

i Case-insensitive pattern matching.

m Treat string as multiple lines.

o Only compile pattern once, even if variables within it change.

s Treat string as single line.

x Use extended regular expressions

Regular Expressions

The patterns used in pattern matching are regular expressions that follow the rules laid out below.

Any single character (or series of characters) matches directly, unless it is a metacharacter with a special meaning. You can cause characters which normally function as metacharacters to be interpreted literally by prefixing them with a "\" (e.g. "\." matches a ".", not any character; "\\" matches a "\"). A series of characters matches that series of characters in the target string, so the pattern *zyxwv* would match "zyxwv" in the target string.

The following metacharacters are as supported:

\ Quote the next metacharacter, including escape sequences (\n, \t etc. apart from \b - see below), ASCII characters ('\nnn' for octal and '\xnn' for hex), and ASCII character controls ('\cx'). '\ n' repeats the part of the 'n'th subpattern that was used to perform the match (not its complete set of rules).

^	Match just the beginning of the string, or with the /m modifier the beginning of any embedded line
.	Match any character (except newline unless the /s modifier is used)
$	Match just the end of the string, or with the /m modifier the end of any embedded line
|	Alternation - to match any one of a set of patterns, usually grouped in brackets.
()	Grouping of subpatterns, numbered automatically left to right by the sequence of their opening parenthesis.
[]	Character class, matching any of the characters in the enclosed list. '^' as the first character in the list negates the expressions - any character *not* in the list.

The following quantifiers are suppported:

*	Match 0 or more times (equivalent to {0,})
+	Match 1 or more times (equivalent to {1,})
?	Match 0 or 1 times (equivalent to {0,1})
{n}	Match exactly n times
{n,}	Match at least n times
{n,m}	Match at least n but not more than m times

Regular expressions also support the following constructs:

Single character matches		Zero width matches	
\w	a "word" character (alphanumeric plus "_")	\b	a word boundary
\W	a non-word character	\B	a non-(word boundary)
\s	a whitespace character	\A	beginning of the string (not embedded newlines)
\S	a non-whitespace character	\Z	end of the string (not embedded newlines)
\d	a digit character	\G	where previous m//g left off

\D a non-digit character

ENVIRONMENT VARIABLES:

The variable ENV is actually a special associative array within Perl that holds the contents of your shell environment variables. You can access any shell environment variable in the same way we accessed the username in Listing 1. For instance, the following commands retrieve the desired PATHand PWD information from a user's environment:

```
$path = $ENV{'PATH'};

$pwd  = $ENV{'PWD'};
```

As a final introductory note regarding associative arrays, notice that the array subscript is enclosed in curly braces. Normal integer-based arrays in Perl are enclosed in square brackets, such as:

```
$current_item = $item_array[100];
```

It's said that associative arrays use fancier brackets than normal arrays because associative arrays are fancier than normal arrays. While I don't know if this statement is 100% accurate, I do find it to be an effective way of remembering the proper subscripting syntax when using Perl arrays.

Printing all of your environment variables in Perl

If you're interested in seeing the contents of all of your environment variables, you can easily print them all out with a foreach command:

```
foreach (sort keys %ENV) {
  print "$_  =  $ENV{$_}\n";
}
```

This example illustrates a fairly common occurrence with associative arrays - how to process each element of an associative array.

The keys function gets all of the keys, or subscripts, out of the specified associative array. In this example, the keys of the associative array %ENVwill be the name of your environment variables. In Listing 1, the key (or subscript) was 'LOGNAME', and the value of the array element $ENV{'LOGNAME'} was fred:

```
$ENV{'LOGNAME'} = "fred";
```

A listing of a few of the keys of the special associative array %ENV are shown in Listing 2.

```
PERL STATEMENT                Key         Value
-----------------------------  ----------  --------------
$ENV{'LOGNAME'} = "fred";       LOGNAME     fred
$ENV{'HOME'}    = "/home/fred"; HOME        /home/fred
```

```
$ENV{'SHELL'}   = "/bin/ksh";     SHELL       /bin/ksh

$ENV{'EDITOR'}  = "vi";           EDITOR      vi
```

Listing 2: This listing shows typical values of several environment variables that would be stored in the associative array %ENV on a Solaris 2.4 system.

When the keys function is run within the foreach loop, it returns only the subscript names LOGNAME, HOME, SHELL, and EDITOR - it does not return the values of each element.

The sort function, which precedes the keys function, sorts the output of the keys command before that output is used by the foreach statement. Given the keys shown in Listing 2, the sorted output would leave the keys in the following order: EDITOR, HOME, LOGNAME, and SHELL.

The foreach statement creates a loop that can be read like this:

> "For each element in the list 'EDITOR, HOME, LOGNAME, and SHELL', do everything enclosed in the following curly braces."

This results in the print statement being run for each key in the %ENV environment variables array.

The $_ variable contains the default pattern space when working with Perl.

Therefore, within the foreach loop, the variable $_ will be assigned the contents of the list of sorted keys, one element at a time. The first time through the loop, the print command

```
print "$_ = $ENV{$_}\n";
```

substitutes for the value of $_, and essentially runs this command:

```
print "EDITOR = $ENV{EDITOR}\n";
```

which results in this output:

```
EDITOR = vi
```

The foreach loop then continues it's processing until every key from the array %ENV has been processed.

USING CGILIB FOR FORMS:

Perl is an excellent language for a variety of tasks, especially those which require text management and data-parsing. Thus, it is well suited for writing code to manage the common gateway interface (CGI) forms which have become the mainstay of world wide web interactive communication via HTML+

cgi-lib.pl is a simple Perl library which is designed to make writing CGI scripts in Perl easy. Some sample forms and scripts are provided here

A form, such as the one here, is just a normal html file. (You can use the "view source" option on your browser to see what the html of that file looks like.) After the user makes selections, the data is processed by a script.

This is a simple form using cgi-lib.pl

This is a sample form which demonstrates the use of the **cgi-lib.pl** library of routines for managing form input.

Pop Quiz:

What is thy name:

What is thy quest:

What is thy favorite color: chartreuse

What is the weight of a swallow: African Swallow or Continental Swallow

What do you have to say for yourself

Press here to submit your query.

Ex:

cgi-lib.pl demo form output

You, , whose favorite color is are on a quest which is , and are looking for the weight of an swallow. And this is what you have to say for yourself:

And here is a list of the variables you entered...

It is often convenient to conflate the form and the script. This can done by having one script file which acts differently depending upon whether it is given parameters or not. (Sophisticated scripts may actually output new forms on the basis of previous forms.) The simple-form.html and simple-form.cgi have been combined into a single combined-form.cgi. The source to this script is

A simple combined form example

This is a sample form which demonstrates the use of the **cgi-lib.pl** library of routines for both generating a form & processing its input.

Pop Quiz:

What is thy name:

What is thy quest:

What is thy favorite color: chartreuse

What is the weight of a swallow: African Swallow or Continental Swallow

What do you have to say for yourself

Press here to submit your query.

A minimalist form and script of only 7 lines can be made this way.

Submit Data:

UNIT IV

POSSIBLE QUESTIONS

(8 MARKS)

1. Explain Arrays concept in PERL with suitable program.
2. Explain looping concept in PERL with suitable program.
3. Describe the concept of perl syntax, variables and strings in detail with suitable program.
4. Explain Pattern Matching in perl with example program
5. Explain Hashes and arguments in perl detail with example program.
6. Describe the following with a sample program in PERL Programming

   Chop

   Length

   Substring
7. Describe about Numbers and Operators in Perl with example program.
8. Briefly describe the concept of cgilib for forms.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
COIMBATORE-641021
Department of Computer Science
III B.Sc(CS) (BATCH 2015-2018)
Open Source Software
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS	**ONE MARK QUESTIONS**	

UNIT IV

S.No	Question	Option1	Option2	Option3	Option4			Answer
1	Perl is a _____	System Program	Programming language	Application Program	Relational Database			Programming language
2	The fourth letter in the LAMP represents _____	PERL	Prototype	Procedure	Program			PERL
3	PERL stands for_____	Practical Extraction and Report Language	Procedural Extraction and Report Language	Practical Extension and Report Language	Procedural Extension and Report Language			Practical Extraction and Report Language
4	Perl orginated as a _____ processing language.	text	file	database	content			text
5	CPAN stands for _____	Comprehensive Procedural Archive Network	Comprehensive Perl Archive Network	Common Perl Archive Network	Common Procedural Archive Network			Comprehensive Perl Archive Network
6	Perl program stored with an _____ extension	.perl	.pl	.per	.prl			.pl
7	____ tells the shell to execute the file	./	.#	.\	.$			./
8	_____ function writes text to standard output	write()	show()	print()	display()			print()
9	Perl scripts are stored as_____	text files	batch files	binary files	executable files			text files
10	In Perl, all statements must end in a ____	:	.	,	;			;
11	The Perl source code file is first compiled into _____	machine code	ascii code	byte code	object code			byte code

12	A _____ scalar variable that holds a single value	array	scalar	hash	associative			scalar
13	String literals can be created with _____	single quotes	double quotes	single quotes and double quotes	without quotes			single quotes and double quotes
14	The _____ can be used to chunk numbers for easy of reading	dollar	underscore	comma	hyphen			underscore
15	What is the base value for octal numerical literal?	10	2	8	16			8
16	Which of the following is not a string literal?	"Hello"	Good'	"Good"	Hello			Hello
17	_____ is an ordered collection of scalars	array	group	list	file			array
18	_____ function adds elements to the right of an array.	add()	push()	insert()	enter()			push()
19	The push() function adds elements to the _____ of an array	specified location	specified value	right	left			right
20	Variables can be declared using _____ function	declare()	decl()	my()	dec()			my()
21	_____ function removes the rightmost element from an array.	pop()	del()	delete()	remove()			pop()
22	_____ are arrays that are indexed not by a number but by a string	Unions	Hashes	Structures	Classes			Hashes
23	Hashes are also known as _____ arrays	descriptive	associative	functioal	Procedural Extension and Report Language			associative
24	Which operator is used in sring concatenation?	.	+	@	&			.

25	Which operator is used in sring replication?	+	-	x	*			x
26	Which of the following ia a compare operator in Perl?	==	<=>	>=	<=			<=>
27	$i=10;$j=$i++; What is the value of i and j?	i=11,j=10	i=10,j=10	i=11,j=11	i=10,j=11			i=11,j=10
28	$a=5;$b=$a--; What is the value of a and b?	a=5,b=4	a=5,b=5	a=4,b=4	a=4,b=5			a=4,b=5
29	$i=10;$j=$++i; What is the value of i and j?	i=11,j=10	i=10,j=10	i=11,j=11	i=10,j=11			i=11,j=11
30	$a=5;$b=$--a; What is the value of a and b?	a=5,b=4	a=5,b=5	a=4,b=4	a=4,b=5			a=4,b=4
31	Regular expressions are otherwise called as____	relational expressions	regexes	logical expressions	assignment expressions			regexes
32	_____ is the new line character in Perl	\n	\t	\r	\f			\n
33	The character class is created using _____	( )	" "	[ ]	{ }			[ ]
34	In Perl, the term function is also referred as _____	method	procedure	class	subroutine			subroutine
35	Arguments are passed into functions through the spcial array____	#_	@_	$_	&_			@_
36	The _____ character is the end-of-file character for the standard input	^D	^E	^S	^F			^E
37	What is the keyboard shortcut for ^D?	Ctrl+D	Ctrl+E	Ctrl+F	Ctrl+S			Ctrl+D
38	_____ function is used to open a file	start()	begin()	open()	create()			open()
39	Which of the following datatypes are preceded by a dollar sign in Perl?	Array	Scalar	Local	Hash			Scalar

40	How to delete a key/value pair to a hash?	using end function	using truncate function	using delete function	using remove function			using delete function
41	Which of the following statement jump to the statement labeled with LABEL?	goto EXPR	goto LABEL	goto NAME	goto ADDRESS			goto LABEL
42	Which of the following operator returns true if the left assignment is stringwise less than or equal to the right assignment?	lt	gt	le	ge			lt
43	Which of the following operator returns a list of values counting from the left value to the right value?	.	x	..	++			..
44	Which of the following function returns epoch time?	local time	gmt time	time	strf time			time
45	Which of the following code create a reference for a variable?	$ref=\$FOO	$ref=\@ARGV	$ref=\%ENV	$ref=\&PrintHash			$ref=\$FOO
46	_____ operator is used to create sequential arrays	arithmetic	relational	logical	range			range
47	What is the range operator?	-	..	--	to			..
48	Which of the following operator returns true if the left assignment is string wise not equal to the right assignment?	lt	ne	le	ge			ne
49	FH stands for_____	File Handles	Field Handle	Folder Handle	Format Handle			File Handle
50	The statement open(FH ,<`abc.txt`) _____	opens the file abc.txt for overwritting	opens the file abc.txt for reading	opens the file abc.txt for rwritting	opens the file abc.txt for appending			opens the file abc.txt for reading

51	Which of the following is used in Perl?	elseif	elsif	elife	eleif			elsif
52	All file handles are named with _____	uppercase letters	lowercase letters	Titlecase letters	upper and lowerccase letters			uppercase letters
53	The > symbol in the open( ) function tells Perl that the file is to be opened in _____ mode	read	write	overwrite	append			write
54	Which operator can be used, when comparing numbers for equivalence?	=	==	both = and ==	<=>			==
55	_____ function sorts the given list	sort()	sortlist()	sorting()	tosort()			sort()
56	unshift() function is similar to _____ function	create()	push()	pop()	my()			push()
57	Which feature of Perl provides code reusability?	function	abstraction	inheritance	encapsulation			inheritance
58	Which of the following shows the warnings in torder t oreduce or avoid errors	- warn	-w	- warnings	- we			-w
59	"The method defines in the parent class will always override the methods defined in the child class" , which is referred as _____	inheritence	polymorphism	encapsultion	abstraction			polymorphism
60	unshift() and shift() functions are oerate _____ of an array	specified location	specified value	rightside	leftside			leftside

UNIT V

SYLLABUS

File Management PERL: - File Handling, ReadingFrom Files, Appending Files, Writing to Files, File Checking, Reading Directories.
Databases PERL: - DBI Module, DBI Connect, DBI Query, MySQL Module, MySQL Connect, MySQL SelectDB, MySQL Query.

FILE MANAGEMENT PERL:

perl-filehandling

Now we shift gears as we introduce file handling. In PERL files are given a name, a handle, basically another way of saying alias. All input and output with files is achieved through filehandling. Filehandles are also a means by one program may communicate with another program.

perl-assigninghandles

A filehandle is nothing more than a nickname for the files you intend to use in your PERL scripts and programs. A handle is a temporary name assigned to a file. A great filehandle is an abreviated version of the filename. The example below illustrates how you will use a file handle in your PERL code.

PERL Code:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #The header
$FilePath = "home/html/myhtml.html"
sysopen(HANDLE, $FilePath, O_RDWR);
printf HANDLE "Welcome to Tizag!";
close (HANDLE);
```

perl-filesandthedie function

The *die* function exists in several programming languages. It is used to kill your scripts and helps pinpoint where/if your code is failing. We use this function as follows.

PERL Code:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #The header

$filepath = "myhtml.html";

sysopen (HTML, '$filepath', O_RDWR|O_EXCL|O_CREAT, 0755) or die "$filepath cannot be
opened.";
printf HTML "<html>\n";
printf HTML "<head>\n";
printf HTML "<title>My Home Page</title>";
printf HTML "</head>\n";
printf HTML "<body>\n";
printf HTML "<p align='center'>Here we have an HTML
page with a paragraph.</p>";
printf HTML "</body>\n";
printf HTML "</html>\n";
close (HTML);
```

Now if for some reason PERL is unable to open or create our file, we will be told. It is good practice to use the die function and we will be using it more as we dive deeper into file handling.

OPENING A FILE

perl-fileopen

Files are opened using the *open* and *sysopen* function. Nothing fancy here at all. Either function may be passed up to 4 arguments, the first is always the file handle discussed earlier, then our file name also known as a URL or filepath, flags, and finally any permissions to be granted to this file.

When opening files as a programmer, there will generally be one of three goals in mind, file creation, appending files, or truncating files.

Create:
Checks to see if the file exists, if not, perl creates a new file.
Append:
Sets the pointer to the end of the file, all output following will be added onto the tail end of the file.
Truncate:
Overwrites your existing file with a new one, this means all data in the old file will be lost.

perl-openafile

The following example will open a previously saved HTML document.

PERL Code:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #The header

$FH = "filehandle";
$FilePath = "myhtml.html";

open(FH, $FilePath, permissions);
or
sysopen(FH, $FileName, permission);
```

Files with special characters or unusual names are best opened by first declaring the URL as a variable. This method removes any confusion that might occur as PERL tries to interpret the code. Tildas in filenames however require a brief character substitution step before they can be placed into your open statements.

perl-filepermissions

File permissions are crucial to file security and function. For instance, in order to function, a PERL file (.pl) must have executable file permissions in order to function on your web server. Also, you may not want all of your HTML files to be set to allow others to write to them or over them. Here's a listing of what to pass to the *open* function when working with file handles.

Shorthand Flags:
Entities Definition

< or r	Read Only Access
> or w	Creates, Writes, and Truncates
>> or a	Writes, Appends, and Creates
+< or r+	Reads and Writes
+> or w+	Reads, Writes, Creates, and Truncates
+>> or a+	Reads, Writes, Appends, and Creates

O_ Flags:

Value	**Definition**
O_RDWR	Read and Write
O_RDONLY	Read Only
O_WRONLY	Write Only
O_CREAT	Create the file
O_APPEND	Append the file
O_TRUNC	Truncate the file
O_EXCL	Stops if file already exists
O_NONBLOCK	*Non-Blocking* usability

PERL Code:

```
#!/usr/bin/perl

print "content-type: text/html \n\n"; #The header
use Fcntl; #The Module

sysopen (HTML, '/home/html/myhtml.html', O_RDWR|O_EXCL|O_CREAT, 0755);
sysopen (HTML, '>myhtml.html');
```

READING A FILE

Reading files

If you want to **read a text file line-by-line** then you can do it as such: `my @lines = <FILE>;` The `<FILE>` operator - where `FILE` is a previously opened filehandle - returns all the unread lines of the text file in list context or a single line in scalar context. Hence, if you had a particularly large file and you wanted to conserve memory you could process it line by line: `while (<FILE>) {`

```
  print $_;
```

}The $_ variable is automatically set for you to the contents of the current line. If you wish you may name your line variable instead: `while (my $line = <FILE>) { ...` will set the `$line` variable to the contents of the current line. The newline character at the end of the line is not removed automatically. If you wish to remove it you can use the `chomp` command. After all lines have been read the `<FILE>` operator will return a false value hence causing the loop to terminate.

There may cases where you need to **read a file only a few characters at a time** instead of line-by-line. This may be the case for binary data. To do just that you can use the `read` command. `open`

```
FILE, "picture.jpg" or die $!;
```

```
binmode FILE;
```

```
my ($buf, $data, $n);
```

```
while (($n = read FILE, $data, 4) != 0) {
```

```
  print "$n bytes read\n";
```

```
  $buf .= $data;
```

```
}
```

```
close(FILE);
```

APPENDING FILES:

Opening a file for writing using the > sign will delete the content of the file if it had any.

If we would like to **append** to the end of the file we use **two greater-than** signs >> as in this example:

```
1.  open(my $fh, '>>', 'report.txt') or die ...
```

Calling this function will open the file for appending. that means the file will remain intact and anything your print() or say() to it will be added to the end.

The full example is this:

```
1.  use strict;
```

```
2.  use warnings;
3.  use 5.010;
4.
5.  my $filename = 'report.txt';
6.  open(my $fh, '>>', $filename) or die "Could not open file
   '$filename' $!";
7.  say $fh "My first report generated by perl";
8.  close $fh;
9.  say 'done';
```

WRITING A FILE:

In order to write to a file, first you need to open the file for writing as follows:

```
open(FH, '>', $filename) or die $!;
```

If the file with filename $filename does not exist, the new file will be created.
Next, you use the print() function to write data into file as follows:

```
print FH $str;
```

You must put space between print(), filehandle FH and $str variable. The $str variable holds data that is written to the file. Notice that if you write to a file that contains content, Perl will truncate its content.
As always, you should close the filehandle when you are no longer use it.

```
close(FH);
```

Putting it all together.

```
#!/usr/bin/perl
use warnings;
use strict;

my $str = <<END;
This is the sample text
```

```
that is used to write to file
END

my $filename = 'c:\temp\test3.txt';

open(FH, '>', $filename) or die $!;

print FH $str;
close(FH);

print "Writing to file successfully!\n";
```

FILE CHECKING:

Before reading from a file or writing to a file, it is important to check if the file exists and readable. In order to perform those tasks, you use Perl file test operators.

The Perl file test operators are logical operators which return `true` or `false` value. For example, to check if a file exists you use `-e` operator as following:

```
#!/usr/bin/perl
use warnings;
use strict;

my $filename = 'c:\temp\test.txt';
if(-e $filename){
  print("File $filename exists\n");
}else{
  print("File $filename does not exists\n");
}
```

The file test operator `-e` accepts a filename or filehandle as an argument.
The following list illustrates the most important Perl file test operators:

- -r: check if the file is readable
- -w: check if the file is writable
- -x: check if the file is executable
- -o: check if the file is owned by effective uid.

- -R: check if file is readable
- -W: check if file is writable
- -X: check if file is executable
- -O: check if the file is owned by real uid.
- -e: check if the file exists.
- -z: check if the file is empty.
- -s: check if the file has nonzero size (returns size in bytes).
- -f: check if the file is a plain file.
- -d: check if the file is a directory.
- -l: check if the file is a symbolic link.
- -p: check if the file is a named pipe (FIFO): or Filehandle is a pipe.
- -S: check if the file is a socket.
- -b: check if the file is a block special file.
- -c: check if the file is a character special file.
- -t: check if the file handle is opened to a tty.
- -u: check if the file has setuid bit set.
- -g: check if the file has setgid bit set.
- -k: check if the file has sticky bit set.
- -T: check if the file is an ASCII text file (heuristic guess).
- -B: check if the file is a “binary” file (opposite of -T).

Using multiple Perl file test operators

If you want to check:

- If a file is a plain file, not directory
- And the file exists
- And the file is readable

You can use the `AND` logical operator in conjunction with file test operators as follows:

```
#!/usr/bin/perl
use warnings;
use strict;

my $filename = 'c:\temp\test.txt';
if(-e $filename && -f _ && -r _ ){
   print("File $filename exists and readable\n");
}
```

Whenever you use the file test operator, Perl will make a new call of stat(), which can be expensive. However, Perl stores the result from the last stat() call to a special filehandle named _, so the subsequent file test operators can use the result that stores in the _ filehandle. Since Perl version 5.9.1 you can stack file test operators as follows:

```
#!/usr/bin/perl
use warnings;
use strict;

my $filename = 'c:\temp\test.txt';
if(-e -f -r $filename){
   print("File $filename exists and readable\n");
}
```

READING DIRECTORIES

Once we have the directory opened we can use the readdir function to read the content of the directory. It can be used either in list or scalar context, just as we were reading from a file in scalar and list context.

In scalar context readdir will always item one, (the 'next') item from the directory. Once we read everything in, it will return undef.

A common way to write it is in a while loop:

```
1.  while (my $thing = readdir $dh) {
2.      say $thing;
3.  }
```

readdir in LIST context

The alternative would be to use readdir in LIST context. For example, to assign it to an array. In that case we might want to iterate over it using a for loop:

```
1.  my @things = readdir $dh;
2.  foreach my $thing (@things) {
3.      say $thing;
4.  }
```

The big difference is that in the second example, all the content of the directory is read in the memory in one statement so it uses more memory. This is much less of an issue here than when we reading the content of a file, as the returned list only contains the names of the things in the directory, which is unlikely to be really big.

Even if we have 100,000 files in a directory, and each one of them has a 10 character long name, it still fits in 1Mb memory.

closedir

Once we are done reading all the things from the directory we can call closedir to officially shut down the connection between the directory handle and the directory on the disk. We don't have to do this though as perl will do it for us when the variable holding the directory handle goes out of scope.

DATABASE PERL

DBI MODULE

perl-dbimodule(s)

PERL is capable of running SQL and MySQL queries including: inserts, selects, updates, deletes, etc through a module termed *DBI*. Often your web host will already have this module as well as *DBD::mysql* already installed. DBI stands for database interface. Any functions associated with DBI should work with all the available SQL platform including: SQL Server, Oracle, DB2, and MySQL.

Before continuing, be sure the following modules are installed:

- DBI
- DBD::mysql

Once they are installed, we can build the introduction to our script by telling PERL to *use* these modules as follows:

dbimodules.pl:

```
#!/usr/bin/perl

# PERL MODULES WE WILL BE USING
use DBI;
use DBD::mysql;
```

Again, these modules allow for us to call upon functions specific to working with a any database platform including MySQL. These modules must be in "use" to ensure proper functionality of our scripts.

perl-dbiconfig

We will be calling on our database, table, and host machine from time to time. We recommend setting up a some variables for your database and table name, so that you can call upon them as you wish throughout this brief tutorial. You may also set up some variables for your user name and password as we will also be needing to connect to your MySQL web host.

dbiconfig.pl:

```
#!/usr/bin/perl

# PERL MODULES WE WILL BE USING
```

```
use DBI;
use DBD::mysql;

# DBI CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
$pw = "password";
```

DBI CONNECT

perl-datasourcename(dsn)

In order to connect to our database platform we first need to know our web server's *data source name*. This information should be readily accessible in your server's documentation. There are four pieces that actively make up a DSN.

- Name of SQL Platform (SQL Server, Oracle, DB2, MySQL, etc).
- Database Name
- Host Name (www.myhost.com)
- Port Number

This information is available from your web host provider and can be defined in PERL as follows:

datasourcename.pl:

```
$dsn = "dbi:SQL Platform:database_name:host_name:port";
```

Since we plan on executing our scripts from our web server through our browser, we can alternatively substitute our host's name with the term *localhost*.

localhost.pl:

```
$dsn = "dbi:SQL_Platform:database_name:localhost:port";
```

perl-dbiconnect

Previously, we had set up a *config* script with some information about our web host and SQL platform including a user name and password. We can now plug all those variables into the connection string and connect to our database.

We can establish a connection with a script like the following.

DBIconnect.pl:

```
#!/usr/bin/perl

# PERL MODULES WE WILL BE USING
use DBI;
use DBD::mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# CONFIG VARIABLES
$platform = "mysql";
$database = "store";
$host = "localhost";
$port = "3306";
$tablename = "inventory";
$user = "username";
$pw = "password";

#DATA SOURCE NAME
$dsn = "dbi:mysql:$database:localhost:3306";


# PERL DBI CONNECT
$DBIconnect = DBI->connect($dsn, $user, $pw);
```

perl-databasehandle

On a side note, we have also created what is known as a *database handle*. Our variable, $DBIconnect, is now the handle which we will have to use each time we wish to execute a query.

We should probably go ahead and shorten up that handle since we will be using it in every query script.

databasehandle.pl:

```
#!/usr/bin/perl

# PERL MODULES WE WILL BE USING
use DBI;
use DBD::mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# CONFIG VARIABLES
$platform = "mysql";
$database = "store";
$host = "localhost";
$port = "3306";
$tablename = "inventory";
$user = "username";
$pw = "password";

#DATA SOURCE NAME
$dsn = "dbi:mysql:$database:localhost:3306";

# PERL DBI CONNECT (RENAMED HANDLE)
$dbstore = DBI->connect($dsn, $user, $pw);
```

The handle has been changed from $DBIconnect, to a more descriptive name.

perl-connection error(s)

An error string variable exists for this module. We can further modify our script with the *die()* function to terminate the script upon connection failure. The error message is usually printed in your web server's error log(s).

databasehandle.pl:

```
#!/usr/bin/perl
```

```
# PERL MODULES WE WILL BE USING
use DBI;
use DBD::mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# CONFIG VARIABLES
$platform = "mysql";
$database = "store";
$host = "localhost";
$port = "3306";
$tablename = "inventory";
$user = "username";
$pw = "password";

#DATA SOURCE NAME
$dsn = "dbi:mysql:$database:localhost:3306";

# PERL DBI CONNECT (RENAMED HANDLE)
$dbstore = DBI->connect($dsn, $user, $pw) or die "Unable to connect: $DBI::errstr\
```

DBI QUERY

perl-dbiquery

Queries must be prepared and then executed. Two lines of code are required for this, first the *prepare()* function and then the *execute()* function.

perl-dbiprepare()

Inside the prepare() function lies the actual SQL query. Essentially the prepare function acts precisely like the console of an SQL platform. If you've been following along, all we need to do is define a variable with a(n) SQL statement. Then create a query handle and run our $connect statement along with the prepare function as outlined below.

The only main difference is that we have to use PERL's escaping characters and we probably have to use them more often.

dbipreparequery.pl:

```
#!/usr/bin/perl

# PERL MODULES WE WILL BE USING
use DBI;
use DBD::mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# CONFIG VARIABLES
$platform = "mysql";
$database = "store";
$host = "localhost";
$port = "3306";
$tablename = "inventory";
$user = "username";
$pw = "password";

# DATA SOURCE NAME
$dsn = "dbi:mysql:$database:localhost:3306";

# PERL DBI CONNECT
$connect = DBI->connect($dsn, $user, $pw);

# PREPARE THE QUERY
$query = "INSERT INTO inventory (id, product, quantity) VALUES (DEFAULT, tomatoes, 4)";
$query_handle = $connect->prepare($query);
```

perl-dbiexecute

Once the query has been prepared, we must execute the command with the execute function. This is accomplished in one final line appended to the code above.

dbiexecutequery.pl:

```
#!/usr/bin/perl

# PERL MODULES WE WILL BE USING
```

```
use DBI;
use DBD::mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# CONFIG VARIABLES
$platform = "mysql";
$database = "store";
$host = "localhost";
$port = "3306";
$tablename = "inventory";
$user = "username";
$pw = "password";

# DATA SOURCE NAME
$dsn = "dbi:$platform:$database:$host:$port";

# PERL DBI CONNECT
$connect = DBI->connect($dsn, $user, $pw);

# PREPARE THE QUERY
$query = "INSERT INTO inventory (id, product, quantity) VALUES (DEFAULT, 'tomatoes',
'4')";
$query_handle = $connect->prepare($query);

# EXECUTE THE QUERY
$query_handle->execute();
```

perl-dbiselectqueries

Select queries fetch results and then return those results in the form of an array. Accessing the results of the array requires first that we bind the columns to variable names. Then we just need to set up a loop to loop through each row and print back the results to our browser.

dbiselectquery.pl:

```
#!/usr/bin/perl
```

```
# PERL MODULES WE WILL BE USING
use DBI;
use DBD::mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# CONFIG VARIABLES
$platform = "mysql";
$database = "store";
$host = "localhost";
$port = "3306";
$tablename = "inventory";
$user = "username";
$pw = "password";

# DATA SOURCE NAME
$dsn = "dbi:mysql:$database:localhost:3306";

# PERL DBI CONNECT
$connect = DBI->connect($dsn, $user, $pw);

# PREPARE THE QUERY
$query = "SELECT * FROM inventory ORDER BY id";
$query_handle = $connect->prepare($query);

# EXECUTE THE QUERY
$query_handle->execute();

# BIND TABLE COLUMNS TO VARIABLES
$query_handle->bind_columns(undef, \$id, \$product, \$quantity);

# LOOP THROUGH RESULTS
while($query_handle->fetch()) {
  print "$id, $product, $quantity <br />";
}
```

Two new functions were introduced in that last example, the *bind_columns* and the *fetch()* functions. Both are fairly self explanatory. Variable names are assigned to our column values via the bind_column function and the fetch() function goes out and fetches the rows matching the query.

MYSQL MODULE

perl-mysql module

MySQL queries and the like can be executed with PERL via the *MySQL*Module. This module should already be installed with your web server if not contact your web host.

As a quick overview, this module installs the necessary functions required to execute MySQL queries using a PERL script. Please take note that this module *only*works with the MySQL platform. Other SQL platforms will require the use of the *DBI*module discussed in our PERL DBI Module lesson.

perl-mysql config

Before we dive head first into the functions, we may want to set up some config variables that we will be calling upon in each script to first connect to our database. Have the following information easily accessible.

- Our Web Host's data source name (DSN)
- User Name for the MySQL Database
- Password for the MySQL Database
- Name of Database
- Name of Table(s)

perlmysqlconfig.pl:

```
#!/usr/bin/perl

# PERL MODULE WE WILL BE USING
use Mysql;

# MySQL CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
```

```
$user = "username";
$pw = "password";
```

A config set-up like this simplifies our connection script and the queries that will be executed later.

MYSQL CONNECT

perl-mysqlconnect

The MySQL module works only with the MySQL platform. We can maintain the same variables from the previous example to connect to MySQL.

perlmysqlconnect.pl:

```
#!/usr/bin/perl

# PERL MODULE
use Mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
$pw = "password";

# PERL MYSQL CONNECT
$connect = Mysql->connect($host, $database, $user, $pw);
```

If this script was run on your web server through a web browser, you should be starring at a blank white screen and all is well.

perl-mysqllistdbs()

Once PERL has established a connection we can execute any of the built in module functions. A great introductory function is the *listdbs* function. This function reads from the MySQL platform and places the name of each database into an array.

listdbs.pl:

```
@databases = $connect->listdbs;
```

We can then loop through this array and print out our results to the browser.

listdbs2.pl:

```
#!/usr/bin/perl

# PERL MODULES
use Mysql;

# MYSQL CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
$pw = "password";

# PERL CONNECT()
$connect = Mysql->connect($host, $database, $user, $pw);

# LISTDBS()
@databases = $connect->listdbs;
foreach $database (@databases) {
        print "$database<br />";
}
```

MYSQL SELECTDB

perl-selectdatabase

In order to perform even the simplest of queries we must first select a database to be working with. Since we have our database name already listed with our config variables, things will be quite simple.

perlmysqlselectdb.pl:

```
#!/usr/bin/perl

# PERL MODULE
use Mysql;

# MYSQL CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
$pw = "password";

# PERL CONNECT()
$connect = Mysql->connect($host, $database, $user, $pw);

# SELECT DB
$connect->selectdb($database);
```

Notice how the syntax requires that we connect to our host each time we perform a function. You will see this with nearly every script we execute. Once we are connected, the sky is the limit as to what queries we can execute.

perl-listtables function

A function exists to list the tables in a database just like the listdbs() function. Use the *listtables()* function to list each table in a database.

listtables.pl:

```
#!/usr/bin/perl

use Mysql;

# HTTP HEADER
```

```
print "Content-type: text/html \n\n";

# MYSQL CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
$pw = "password";

# PERL MYSQL CONNECT()
$connect = Mysql->connect($host, $database, $user, $pw);

# SELECT DB
$connect->selectdb($database);

# LISTTABLES()
@tables = $connect->listtables;

# PRINT EACH TABLE NAME
@tables = $connect->listtables;
foreach $table (@tables) {
        print "$table<br />";
}
```

The database is defined when we run the $connect variable. To change the script to a different database simply run a new selectdb() function or change the $database variable.

MYSQL QUERY

perl-mysqlquery

Executing a query using the MySQL module is a two step process - very straight forward. We define a query in the form of a scalar variable then call upon that variable using our connection script and the *query* function.

perlmysqlquery.pl:

```
# DEFINE A MySQL QUERY
$myquery = "INSERT INTO $tablename
```

```
(id, product, quantity)
VALUES (DEFAULT,'pineapples','15')";

# EXECUTE THE QUERY FUNCTION
$execute = $connect->query($myquery);
```

perl-mysqlinsertquery

Here we introduce the *affectedrow()* function along with the *insertid()* function. You can probably guess what the affected rows function does but insertid is unique. Inserid() returns the 'id' of the last inserted row, that is it will return an id if you have an id field set up to auto-increment in your MySQL table.

perlinsertquery.pl:

```
#!/usr/bin/perl

use Mysql;

print "Content-type: text/html \n\n";

# MYSQL CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
$pw = "password";

# PERL MYSQL CONNECT()
$connect = Mysql->connect($host, $database, $user, $pw);

# SELECT DB
$connect->selectdb($database);

# DEFINE A MySQL QUERY
$myquery = "INSERT INTO
$tablename (id, product, quantity)
VALUES (DEFAULT,'pineapples','15')";
```

```
# EXECUTE THE QUERY FUNCTION
$execute = $connect->query($myquery);

# AFFECTED ROWS
$affectedrows = $execute->affectedrows($myquery);

# ID OF LAST INSERT
$lastid = $execute->insertid($myquery);

print $affectedrows."<br />";
print $lastid."<br />";
```

These functions could be run without defining them as scalar variables as well.

perl-mysqlselectquery

Queries that use the *SELECT* clause are a little more exciting. Here we introduce two new functions, the *numrows()* function and the *numbfields()* function. Both of these do exactly as they say, one fetches the number of rows returned with as the query executes while the other fetches the number of fields returned.

easyselectfunctions.pl:

```
#!/usr/bin/perl

use Mysql;

# HTTP HEADER
print "Content-type: text/html \n\n";

# MYSQL CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
$pw = "password";

# PERL MYSQL CONNECT()
$connect = Mysql->connect($host, $database, $user, $pw);
```

```
# SELECT DB
$connect->selectdb($database);

# DEFINE A MySQL QUERY
$myquery = "SELECT * FROM $tablename";

# EXECUTE THE QUERY
$execute = $connect->query($myquery);

$rownumber = $execute->numrows();
$fieldnumber = $execute->numfields();

# PRINT THE RESULTS
print $rownumber."<br />";
print $fieldnumber."<br />";
```

Two numbers should be printed to your web browser.

perl-mysql*fetchrow()*

The *fetchrow()* function does exactly as it says it does, it goes out and fetches a row that matches your MySQL Query. An array is returned and each element represents a column value for the fetched row. If the query is intended to return multiple rows, *fetchrow()* must be called again and again. This is easily accomplished with a *while* loop.

fetchrow.pl:

```
#!/usr/bin/perl

use Mysql;

print "Content-type: text/html \n\n";

# MYSQL CONFIG VARIABLES
$host = "localhost";
$database = "store";
$tablename = "inventory";
$user = "username";
```

```
$pw = "password";

# PERL MYSQL CONNECT()
$connect = Mysql->connect($host, $database, $user, $pw);

# SELECT DB
$connect->selectdb($database);

# DEFINE A MySQL QUERY
$myquery = "SELECT * FROM $tablename";

# EXECUTE THE QUERY FUNCTION
$execute = $connect->query($myquery);

# HTML TABLE
print "<table border='1'><tr>
<th>id</th>
<th>product</th>
<th>quantity</th></tr>";

# FETCHROW ARRAY

while (@results = $execute->fetchrow()) {
        print "<tr><td>"
        .$results[0]."</td><td>"
        .$results[1]."</td><td>"
        .$results[2]."</td></tr>";
}

print "</table>";
```


UNIT V

POSSIBLE QUESTIONS

(8 MARKS)

1. Describe file management in detail with suitable example.
2. Describe about DBI Module, DBI Connect and DBI Query in detail.
3. Explain the following in Perl Programming
    i. i)Reading from file
    ii. ii)Appending a file
4. Describe MySQL module, MySQL Connect and MySQl SelectDB.
5. Explain about the concept of writing to files, file checking and reading directories in detail.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
COIMBATORE-641021
Department of Computer Science
III B.Sc(CS) (BATCH 2015-2018)
Open Source Software
PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS **ONE MARK QUESTIONS**

UNIT V

S.No	Question	Option1	Option 2	Option3	Option4			Answer
1	The father of MySQL is_____	Micheal Widenious	Bill Joy	Bill Gates	Stephanie Wall			Micheal Widenious
2	MySQL development Project has made its source code available under the terms of _____	LGPL	GPL	MPL	BSD			GPL
3	The letter M in the LAMP represents _____	method	memory	machine	MySQL			MySQL
4	MySQL comes with as standard with client libraries for _____	Java only	C	Java and C only	Perl, PHP and C			C
5	_____ is an open source standard query language database that is fast, reliable for applications of any size.	MySQL	SQL Server	Oracle 10g	Oracle 11g			MySQL
6	SQL stands for _____	Standard Query Language	System Query Language	Structured Query Language	Service Query Language			Structured Query Language
7	DBI stands for_____	DataBase Independent Interface	DataBase Information Interface	Design Independent Interface	Design Information Interface			DataBase Independent Interface
8	API stands for_____	Application Procedural Information	Application Procedural Interface	Application Programming Interface	Application Programming Information			Application Programming Interface

9	_____ is a container for related tables.	database	datawarehouse	datacentre	datamart			database
10	A table is a collection of_____	database	fields	rows	column			rows
11	Each row holding data for _____ record	one	two	three	four			one
12	Each record containing chunks of information called_____	rows	fields	cells	values			fields
13	BLOB stands for _____	Binary Large Object	Basic Logical Object	Binary Logical Object	Basic Large Object			Binary Large Object
14	Which of the following command is used to display the currently available databases?	SHOW DATABASES	DISPLAY DATABASES	PRINT DATABASES	VIEW DATABASES			SHOW DATABASES
15	_____ command is used to connect the database to MySQL	CREATE	USE	CONNECT	CONNECTDB			USE
16	The _____ command gives the information about the fields in a table	DESCRIBE	SHOW	DISPLAY	VIEW			DESCRIBE
17	DESCRIBE Command can be abbreviated as_____	DESCRIBE	DESC	DEBE	DCBE			DESC
18	_____ command used to change the value in an existing record.	INSERT	ALTER	DESCRIBE	UPDATE			UPDATE
19	The keyword SET is used with the command_____	UPDATE	SELECT	DELETE	ALTER			UPDATE
20	_____ method causes the Perl script to connect to the MySQL database.	combine()	connect()	allow()	redirect()			connect()
21	_____ method allowing the perl script and database to properly shutdown the connection	disallow()	shutdown()	disconnect()	terminate()			disconnect()
22	Which symbol in SELECT command shows values for all fields in the table?	*	+	@	#			*
23	DESCRIBE Command is equvalant to _____	SHOW DATABASES	SHOW TABLES	SHOW ROWS	SHOW COLUMNS			SHOW COLUMNS

24	DBH stands for_____	DataBase Handler	DataBase Holder	DataBase Hoster	DataBase Handle			DataBase Handle
25	If connect() return false, the script dies printing the error string returned by the _____ method	error()	errormsg()	errstr()	errorstring()			errstr()
26	The _____ method returns alist of data for the next row of data that is returned by the SELECT query	fetchdata()	fetchrecord()	fetchlist()	fetchrow()			fetchrow()
27	MySQL runs on which OS?	Linux and Mac Operating System-X only	Any Operating System at all	Unix, Linux, Windows and other	Unix and Linux only			Unix, Linux, Windows and other
28	To remove duplicate rows from the result of a SELECT use the _____ keyword.	NO DUPLICATE	UNIQUE	DISTINCT	REMOVE DUPLICATE			DISTINCT
29	Which of the following can add a row to a table?	ADD	INSERT	UPDATE	ALTER			INSERT
30	To use MySQL on your computer, it need _____	FTP and Telnet	Some sort of client program to access the databases	a browser	Perl, PHP or Java			Some sort of client program to access the databases
31	In a LIKE clause you could ask for any value ending in "ter" by writing_____	LIKE %ter	LIKE &ter	LIKE *ter	LIKE ^ter			LIKE %ter
32	A NULL Value is treated as_____	balnk	zero	NULL value	No value			NULL value
33	MySQl is a _____	programming language	web designng language	technique for writing reliable program	Relational Database Management System			Relational Database Management System
34	which function used to get the current time in MySQL?	getTime()	Time()	Now()	showTime()			Now()

35	Which of the following is not a valid aggregate function?	COUNT	MIN	MAX	COMPUTE			COMPUTE
36	WhatSQL clause is used to restrict the rows returned by a query?	AND	WHERE	HAVING	FROM			WHERE
37	How much character are allowed to create database name?	55	72	64	40			64
38	Which of the following command is used creates a database?	CREATE DB student	CREATE DATABASE student	CREATE DBASE student	CREATE student DATABASE			CREATE DATABASE student
39	Which one will delete the table data as well as table stucture?	TRUNCATE	DROP	REMOVE	DELETE			DROP
40	The main MySQL program that does all the data handling is called_____	mysql.exe	mysql	mysqld	httpd			mysqld
41	A SELECT command without a WHERE clause returns ____	all records from a table	no records	SELECT is invalid without a WHERE clause	all records from a table that match the previous WHERE clause			all records from a table
42	Which statement is used to access an existing database?	USE database.name	USE databasename	USE	USE DBASE			USE
43	The MySQL command line tool format the results in which of the following format?	Rectangle	Square	Sphere	Circle			Rectangle
44	The "MySQL command line tool" formats are bounded by _____	+-*	+-|	+-/	+-}			+-|
45	What kind of replication is supported by the MySQL server?	multiple master replication	master to slave replication	single file based clustering	MySQL doesn't support replication			master to slave replication

46	Commands passed to the MySQL daemon are written in_____	programming language	web designng language	structured suery Languag e	machine language			structured suery Languag e
47	Which of the following is not a valid name for a column?	insert	name	age	gender			insert
48	Which of these commands will delete a table called "xxx"?	DROP xxx	DROP TABLE xxx	DELETE xxx	DELETE TABLE xxx			DROP TABLE xxx
49	Which of the following is not supported by MySQL?	temporary tables	table joining	stored procedures	regular expression			stored procedures
50	_____ command is used to undo a GRANT privilege	REVOKE	UNDO	UNGRANT	ROLLBACK			REVOKE
51	How many distinct, different values can hold in an enum field?	255	7	65535	2			65535
52	Which of the following command is not availble?	REVOKE	FETCH	UPDATE	SELECT			FETCH
53	Which of these field datatyppes would be best to hold a film title?	longblob	tinytext	mediumtext	longtext			tinytext
54	Which of these field datatyppes would be best to hold a .jpg file?	char	nchar	text	blob			blob
55	On executing DELETE command, if you get an error "foreign key constraint"- What does it imply?	Foreign key is not defined	Table is empty	Connectivity issue	Data is present in other table			Data is present in other table
56	How much storage space does DATETIME require?	4 Bytes	2 Bytes	8 Bytes	1 Byte			8 Bytes
57	User() function returns the current user's username and _____	password	hostname	both password and hostname	database name associated with that user			hostname
58	What is a primary Key_____	used to uniquely identify a row	alias for candidate key	used to identify a column	alias for foreign key			used to uniquely identify a row
59	A view is nothing but a _____ view or a stored query.	static	dynamic	virtual	real			virtual

60	_____ aggregate function is used to get the number of records or rows in a table.	COUNTROWS()	COUNT()	SUM()	NUMBER()			COUNT()

Register Number_____

[15CSU603A]

KARPAGAM ACADEMY FOR HIGHER EDUCATION

Coimbatore - 641021.

(For the candidates admitted from 2015 onwards)

FIRST INTERNAL EXAMINATION, JANUARY 2018

Sixth Semester

COMPUTER SCIENCE

OPEN SOURCE SOFTWARE

Date & Session : .01.2018 **Class: III B.Sc CS**

Maximum : 50 Marks **Duration: 2 Hours**

PART-A (20 X 1 = 20 Marks)

(Answer ALL the Questions)

1. OSS stands for _____
   - **a) Open Source Software**
   - b) Open Standard Software
   - c) Open Standard Service
   - d) Open Source Service

2. Which of the following is not an example of open source software?
   - a) Linux
   - **c) Windows**
   - b) Mozilla
   - d) Apache

3. The human readable code of the program is known as _____
   - **a) source code**
   - b) byte code
   - c) machine code
   - d) object code

4. _____ is a proprietary software made available free of charge
   - a) Free software
   - **c) Freeware**
   - b) Open source
   - d) Malware

5. BSD stands for _____
   - **a) Berkeley Software Distribution**
   - b) Berkeley Source Distribution
   - c) Berkeley Software Destruction
   - d) Berkeley Source Destruction

6. "Wikipedia is an example of_____ software
   - a) freeware
   - c) proprietary
   - **b) open source**
   - d) free

7. BSD licenses also referred as_____
   - **a) BSD style**
   - b) BSD unique
   - c) BSD source
   - d) BSD type

8. FSF stands for_____
   - **a) Free Software Foundation**
   - b) Free System Foundation
   - c) Free Service Foundation
   - d) Free Source Foundation

9. Which of the following license the software cannot be mixed with non-free software

   a)MIT b)LGPL c)**GPL** d)BSD

10. _____ is a form of intellectual property , applicable to certain forms creative works
    - a) License
    - c) copyleft
    - b) patent
    - d) **copyright**

11.The development of open source hardware was initiated in _____

a) 2004 b)2000 c)**2002** d)2001

12._____ is a software that is floated to see if there is an interest

a) Vaporware
b) Freeware
c) Free Software
d) Open Source Software

13.Richard's Stallman organization is called _____

a) Free Software Foundation
b) Open Source Software Foundation
c) Close Source Software Foundation
d) Apache Software Foundation

14.Which of the following is not true for open source software?

a) It is owned by a person
b) It supports distributed development
c) It supports collaborative development
d) Its code is available for all

15.A software that has both proprietary license and open source license is called _____.

a) multiple licensed software
b) uncontrolled software
c) dual licensed software
d) mixed licensed software

16.Dual licensing works for _____.

a) work having many contributors
b) collaboratively developed software
c) single, well defined owner of a work
d) open source software following development strategy

17.Proprietary licenses _____ for that consideration.

a) have some rights and pay a fee
b) have no rights and pay little fee
c) have some rights and pay no fee
d) have no rights and pay no fee

18.Ownership applies to _____.

a) tangible properties only
b) both tangible and intangible properties
c) intangible properties only
d) software only

19.The software in wide distribution that has a version number less than _____

a) 3.0 b)2.0 c)**1.0** d)0.5

20.Which of the following is proprietary?

a) OpenOffice b)**Oracle** c)MySQL d)Postgres

PART-B (3 X 10 = 10 Marks)

(Answer ALL the Questions)

21. a) Explain about open source software and list out the examples of OSD complaint licenses.

Open source software is computer software that has a source code available to the general public for use as is or with modifications. This software typically does not require a license fee. There are open source software applications for a variety of different uses such as office automation, web design, content management, operating systems, and communications. 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. These criteria include the right to:

- Redistribute the software without restriction;
- Access the source code;
- Modify the source code; and
- Distribute the modified version of the software.

In contrast, creators of proprietary software usually do not make their source code available to others to modify. When considering the advantages of open source software you should consider the open source product itself. Open source products vary in quality. OSS software does not come with phone support or personalized e-mail support. However, there are commercial service providers who will provide support. If you need a lot of support, consider whether the overall costs of using an open source product will be higher than that of a proprietary product.

Nobody owns or controls the term “Open Source”, as it was deemed too broad and descriptive to be a trademark under US Law. However, in general use, open source software is software distributed under terms that comply with the Open Source Definition (OSD). The OSD is a document maintained by the Open Source Initiative(OSI).

Furthermore, it is eligible to bear the OSI Certified certification mark (Perens, 1999; Open Source Initiative, 2001a). According to the OSI, the OSI certified certification mark “applies to software, not to licenses” (Open Source Initiative, 2001a). However, in practice, the OSD has been used mainly as a licensing standard, and the OSI maintains a list of OSD-compliant licenses. The majority of OSS products in circulation are self-certified (they are distributed under the terms of a previously approved license, and are thus implicitly trusted to implement it properly) and are not evaluated by the OSI on a product-byproduct basis. Developers may also, of course, submit a new license for OSI approval.

Either way, the OSI Certified mark is used by attaching one of two notices to the software product, namely, "This software is OSI Certified Open Source Software. OSI Certified is a certification mark of the Open Source Initiative" or, simply, "OSI Certified Open Source Software" (Open Source Initiative, 2001a).

According to the Open Source Definition (Open Source Initiative, 2001b): Open Source doesn't just mean access to the source code. The distribution terms of open source software must comply with the following criteria:

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

2. Source Code: The program must include source code, and must follow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost- preferably, downloading via the internet without charge. The source code must be the preferred form in which a programmer would modify the program.

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

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

5. No Discrimination against Persons or Groups: The license must not discriminate against any person or group of persons.

6. No Discrimination against Fields of Endeavor: The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

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

8. License must not be specific to a product: The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License must not contaminate other software: The license must not place restrictions on other software that is distributed along with the licensed software.

For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

EXAMPLES OF OSD-COMPLIANT LICENSES

The Open Source Initiative (2001c) provides a list of licenses that have been reviewed

and found to be compliant with the OSD. There are 21 licenses on the list, namely

The GNU General Public License (GPL)

The GNU Lesser Public License (LGPL)

The Berkeley Software Distribution (BSD) License

The MIT License

The Artistic License

The Mozilla Public License (MPL)

The Qt Public License (QPL)

The IBM Public License

The MITRE Collaborative Virtual Workspace License (CVW License)

The Ricoh Source Code Public License

The Python License

The zlib/libpng license

The Apache Software License

The Vovida Software License

The Sun Internet Standards Source License (SISSL)

The Intel Open Source License

The Jabber Open Source License

The Nokia Open Source License

The Sleepycat License

The Nethack General Public License

Since all of these licenses conform to the OSD, we will limit our comments to the more distinctive qualities of the most widely used licenses. The GPL and LGPL were created by Richard Stallman's Free Software Foundation (FSF) and, in fact, predate the coining of the term Open Source. There is an enormous amount of GPL-licenses software in circulation- 11723 independent projects hosted at the SourceForge website alone and the FSF itself has produced over 170 mature products, collectively referred to as the GNU Project.

[OR]

b) Write a short note on

(i) Linux (ii)Apache (iii)Mozilla

LINUX

Linux is the best-known and most-used open source operating system. As an operating system, Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware.

For the purposes of this page, we use the term "Linux" to refer to the Linux kernel, but also the set of programs, tools, and services that are typically bundled together with the Linux kernel to provide all of the necessary components of a fully functional operating system. Some people, particularly members of the Free Software Foundation, refer to this collection as GNU/Linux, because many of the tools included are GNU components. However, not all Linux installations use GNU components as a part of their operating system. Android, for example, uses a Linux kernel but relies very little on GNU tools.

How does Linux differ from other operating systems?

In many ways, Linux is similar to other operating systems you may have used before, such as Windows, OS X, or iOS. Like other operating systems, Linux has a graphical interface, and types of software you are accustomed to using on other operating systems, such as word processing applications, have Linux equivalents. In many cases,

the software's creator may have made a Linux version of the same program you use on other systems. If you can use a computer or other electronic device, you can use Linux.

But Linux also is different from other operating systems in many important ways. First, and perhaps most importantly, Linux is open source software. The code used to create Linux is free and available to the public to view, edit, and—for users with the appropriate skills—to contribute to.

Linux is also different in that, although the core pieces of the Linux operating system are generally common, there are many distributions of Linux, which include different software options. This means that Linux is incredibly customizable, because not just applications, such as word processors and web browsers, can be swapped out. Linux users also can choose core components, such as which system displays graphics, and other user-interface components.

What is the difference between Unix and Linux?

We may have heard of Unix, which is an operating system developed in the 1970s at Bell Labs by Ken Thompson, Dennis Ritchie, and others. Unix and Linux are similar in many ways, and in fact, Linux was originally created to be similar to Unix. Both have similar tools for interfacing with the systems, programming tools, filesystem layouts, and other key components. However, Unix is not free. Over the years, a number of different operating systems have been created that attempted to be "unix-like" or "unix-compatible," but Linux has been the most successful, far surpassing its predecessors in popularity.

Who uses Linux?

We are probably already using Linux, whether you know it or not. Depending on which user survey you look at, between one- and two-thirds of the webpages on the Internet are generated by servers running Linux.

Companies and individuals choose Linux for their servers because it is secure, and you can receive excellent support from a large community of users, in addition to companies like Canonical, SUSE, and Red Hat, which offer commercial support.

Many of the devices you own probably, such as Android phones, digital storage devices, personal video recorders, cameras, wearables, and more, also run Linux. Even your car has Linux running under the hood.

Who "owns" Linux?

By virtue of its open source licensing, Linux is freely available to anyone. However, the trademark on the name "Linux" rests with its creator, Linus Torvalds. The source code for Linux is under copyright by its many individual authors, and licensed under the GPLv2 license. Because Linux has such a large number of contributors from across multiple decades of development, contacting each individual author and getting them to agree to a new license is virtually impossible, so that Linux remaining licensed under the GPLv2 in perpetuity is all but assured.

How was Linux created?

Linux was created in 1991 by Linus Torvalds, a then-student at the University of Helsinki. Torvalds built Linux as a free and open source alternative to Minix, another Unix clone that was predominantly used in academic settings. He originally intended to name it "Freax," but the administrator of the server Torvalds used to distribute the original code named his directory "Linux" after a combination of Torvalds' first name and the word Unix, and the name stuck.

Most of the Linux kernel is written in the C programming language, with a little bit of assembly and other languages sprinkled in. If you're interested in writing code for the Linux kernel itself, a good place to get started is in the Kernel Newbies FAQ, which will explain some of the concepts and processes you'll want to be familiar with.

But the Linux community is much more than the kernel, and needs contributions from lots of other people besides programmers. Every distribution contains hundreds or

thousands of programs that can be distributed along with it, and each of these programs, as well as the distribution itself, need a variety of people and skill sets to make them successful, including:

- Testers to make sure everything works on different configurations of hardware and software, and to report the bugs when it does not.
- Designers to create user interfaces and graphics distributed with various programs.
- Writers who can create documentation, how-tos, and other important text distributed with software.
- Translators to take programs and documentation from their native languages and make them accessible to people around the world.
- Packagers to take software programs and put all the parts together to make sure they run flawlessly in different distributions.
- Evangelists to spread the word about Linux and open source in general.
- And of course developers to write the software itself.

The GNU/Linux distributions that are entirely free as in freedom. All of the distributions that follow are installable to a computer's hard drive; most can be run live.

The Free Software Foundation recommends and endorses these GNU/Linux distros, although we do not try to judge or compare them based on any criterion other than freedom; therefore, we list them in alphabetical order. We encourage you to read these brief descriptions and to consult their respective web sites and other information to choose the one best for you.

These distros are ready-to-use full systems whose developers have made a commitment to follow the Guidelines for Free System Distributions. This means these distros will include, and propose, exclusively free software. They will reject nonfree applications, nonfree programming platforms, nonfree drivers, nonfree firmware “blobs”, nonfree games, and any other nonfree software, as well as nonfree manuals or documentation.

If one of these distros ever does include or propose anything nonfree, that must have happened by mistake, and the developers are committed to removing it. If you find

nonfree software or documentation in one of these distributions, you can report the problem, and earn GNU Bucks, while we inform the developers so they can fix the problem.

Fixing freedom bugs is an ethical requirement for listing a distro here; therefore, we list only distros with a development team that has told us it will remove any nonfree software that might be found in them. Usually the team consists of volunteers, and they don't make legally binding commitments to users; but if we find out a distro is not properly maintained, we will de-list it.

We hope the other existing GNU/Linux distributions will become entirely free software so that we can list them here. If you wish to improve the state of free distros, helping to develop an existing free distro contributes more than starting a new one.

All of the distributions that follow are installable to a computer's hard drive; most can be run live. Not all hardware works in the free world; each distro's site should say which hardware it supports.

APACHE

The **Apache Software Foundation** (**ASF**) is an American non-profit corporation (classified as 501(c)(3) in the United States) to support Apache software projects, including the Apache HTTP Server. The ASF was formed from the Apache Group and incorporated in Delaware, U.S., in June 1999.

The Apache Software Foundation is a decentralized open source community of developers. The software they produce is distributed under the terms of theApache License and is free and open-source software (FOSS). The Apache projects are characterized by a collaborative, consensus-based development process and an open and pragmatic software license.

Each project is managed by a self-selected team of technical experts who are active contributors to the project. The ASF is a meritocracy, implying that membership of the foundation is granted only to volunteers who have actively contributed to Apache projects. The ASF is considered a second generation open-source organization, in that commercial support is provided without the risk of platform lock-in.

Among the ASF's objectives are: to provide legal protection[4] to volunteers working on Apache projects; to prevent the *Apache* brand name from being used by other organizations without permission.

The ASF also holds several ApacheCon conferences each year, highlighting Apache projects and related technology.

The history of the Apache Software Foundation is linked to the Apache HTTP Server, development beginning in February 1993. A group of eight developers started working on enhancing the NCSA HTTPd daemon. They came to be known as the Apache Group. On March 25, 1999, the Apache Software Foundation was formed.The first official meeting of the Apache Software Foundation was held on April 13, 1999, and by general consent that the initial membership list of the Apache Software Foundation, would be: Brian Behlendorf, Ken Coar, Miguel Gonzales, Mark Cox, Lars Eilebrecht, Ralf S. Engelschall, Roy T. Fielding, Dean Gaudet, Ben Hyde, Jim Jagielski, Alexei Kosut, Martin Kraemer, Ben Laurie, Doug MacEachern, Aram Mirzadeh, Sameer Parekh, Cliff Skolnick, Marc Slemko, William (Bill) Stoddard, Paul Sutton, Randy Terbush and Dirk-Willem van Gulik. After a series of additional meetings to elect board members and resolve other legal matters regarding incorporation, the effective incorporation date of the Apache Software Foundation was set to June 1, 1999.

The name 'Apache' was chosen from respect for the Native American Apache Nation, well known for their superior skills in warfare strategy and their inexhaustible endurance. It also makes a pun on "a patchy web server"—a server made from a series of patches—but this was not its origin. The group of developers who released this new software soon started to call themselves the "Apache Group".

Oracle, IBM, and the Apache Software Foundation jointly announced last week that OpenOffice.org would become an official Apache project. OpenOffice.org is an important piece of free software, and many of its supporters suggest that this change will give them more control over the project's future direction. However, users and contributors should be aware that, as part of this transition, it will become easier for proprietary software developers to distribute OpenOffice.org as nonfree software.

All Apache projects are distributed under the terms of the Apache License. This is a non-copyleft free software license; anybody who receives the software can distribute it to others under nonfree terms. Such a licensing strategy represents a significant policy change for OpenOffice.org. Previously, the software was distributed under the terms of the GNU Lesser General Public License (LGPL). The LGPL is a weak copyleft license, so programs that merely link to the software can be released under nonfree terms, but the software covered by the LGPL must always be released, along with its source code, under the LGPL's terms. Free software developers are clearly comfortable with a partial copyleft when it's appropriate; in numerous surveys of free software projects, the LGPL is commonly listed as the second-most popular license (after the GNU General Public License), or else follows close behind.

While we do recommend the Apache License in specific situations, we do not believe it is the best choice for software like OpenOffice.org. This situation calls for copyleft, because the gains free software stands to make from a non-copyleft license don't justify giving a handout to proprietary software developers.

Fortunately, there's a ready alternative for people who want to work with a productivity suite that does more to protect their freedom: LibreOffice. Anybody who's comfortable with OpenOffice.org will find a familiar interface and feature set in LibreOffice, because it was originally based on the same source code. Since September 2010, numerous contributors have been working to improve the software, and the project's legal steward, The Document Foundation, is committed to keeping it licensed under the LGPL.

LibreOffice's commitment to user freedom does not end at the license of its source code. Like OpenOffice.org, the software's built-in extension manager makes it easy to add new features, but unlike OpenOffice.org, its extension database only lists add-ons that are under a free license. OpenOffice.org points to a database that includes proprietary extensions, and doesn't always provide clear licensing information. This approach to extensions risks turning free software into a platform for the development and promotion of proprietary extras.

Anybody who plans to use or contribute to one of these productivity suites should understand how these policies affect them, and consider which better complement their own goals. While both pass the most important test of being free software, we recommend LibreOffice because its policies do significantly more to promote the cause of free software.

MOZILLA

Mozilla is a free-software community created in 1998 by members of Netscape. The Mozilla community uses, develops, spreads and supports Mozilla products, thereby promoting exclusively free software and open standards, with only minor exceptions.The community is supported institutionally by the Mozilla Foundation and its tax-paying subsidiary, the Mozilla Corporation.

Mozilla's products include the Firefox web browser, Thunderbird e-mail client, Firefox OS mobile operating system, Bugzilla bug tracking system,Gecko layout engine and others. During 2017, Mozilla acquired Pocket, a "read-it-later-online" service.

Firefox is a web browser, and is Mozilla's flagship software product. It is available in both desktop and mobile versions. Firefox uses the Gecko layout engine to

render web pages, which implements current and anticipated web standards. As of late 2015, Firefox had approximately 10-11% of worldwide usage share of web browsers, making it the 4th most-used web browser.

Firefox began as an experimental branch of the Mozilla codebase by Dave Hyatt, Joe Hewitt and Blake Ross. They believed the commercial requirements of Netscape's sponsorship and developer-driven feature creep compromised the utility of the Mozilla browser.[48] To combat what they saw as the Mozilla Suite's software bloat, they created a stand-alone browser, with which they intended to replace the Mozilla Suite.

Firefox was originally named *Phoenix* but the name was changed so as to avoid trademark conflicts with Phoenix Technologies. The initially-announced replacement, *Firebird*, provoked objections from the Firebird project community. The current name, Firefox, was chosen on February 9, 2004.

Firefox OS

Firefox OS was an open source operating system in development by Mozilla that aims to support HTML5 apps written using "open Web" technologies rather than platform-specific native APIs. The concept behind Firefox OS is that all user-accessible software will be HTML5 applications, that use Open Web APIs to access the phone's hardware directly via JavaScript.

Some devices using this OS include Alcatel One Touch Fire, ZTE Open, and LG Fireweb.

Thunderbird

Thunderbird is a free, open source, cross-platform email and news client developed by the volunteers of the Mozilla Community.

On July 16, 2012, Mitchell Baker announced that Mozilla's leadership had come to the conclusion that on-going stability was the most important thing for Thunderbird and that innovation in Thunderbird was no longer a priority for Mozilla. In that update Baker also suggested that Mozilla had provided a pathway for community to innovate around Thunderbird if the community chooses.

SeaMonkey

SeaMonkey (formerly the Mozilla Application Suite) is a free and open source cross platform suite of Internet software components including a web browser component, a client for sending and receiving email and Usenet newsgroup messages, an HTML editor (Mozilla Composer) and the ChatZilla IRC client.

On March 10, 2005, the Mozilla Foundation announced that it would not release any official versions of Mozilla Application Suite beyond 1.7.x, since it had now focused on the standalone applications Firefox and Thunderbird. SeaMonkey is now maintained by the SeaMonkey Council, which has trademarked the SeaMonkey name with help from the Mozilla Foundation.The Mozilla Foundation provides project hosting for the SeaMonkey developers.

Bugzilla

Bugzilla is a web-based general-purpose bug tracking system, which was released as open source software by Netscape Communications in 1998 along with the rest of the Mozilla codebase, and is currently stewarded by Mozilla. It has been adopted by a variety of organizations for use as a bug tracking system for both free and open source software and proprietary projects and products, including the Mozilla Foundation, the Linux kernel, GNOME, KDE, Red Hat, Novell, Eclipse andLibreOffice.

Mozilla is open source and free software – any person or company is free to:

- run the program, for any purpose;
- study how the program works, and adapt it to their needs;
- redistribute copies at will;
- improve the program, and distribute the altered version.

All the source code for Mozilla is available under the Mozilla and Netscape Public Licenses, which are accepted as free software licenses by the Free Software Foundation.

The spirit of the MPL is that you are free to use Mozilla code in your applications and products – including proprietary products – provided that you make available any modifications you make to the actual Mozilla code base itself.

With free software, your business is not locked into the products of one company. You are free to control your own future.

22. a) Describe the concept of Berkeley software distribution in detail.

Berkeley Software Distribution (BSD) was a Unix operating system derivative developed and distributed by the Computer Systems Research Group (CSRG) of the University of California, Berkeley, from 1977 to 1995. Today the term "BSD" is often used non-specifically to refer to any of the BSD descendants which together form a branch of the family of Unix-like operating systems. Operating systems derived from the original BSD code remain actively developed and widely used.

Short for Berkeley Software Distribution, BSD is a Unix-like operating system first introduced in late 1977. Originally titled 1BSD, it was developed at the Computer System Research Group (CSRG) of the University of California at Berkeley. Today, BSD comes in various flavors such as BSDi Internet Server (BSD/OS), FreeBSD, NetBSD, and OpenBSD below is a brief introduction to each of these flavors of BSD.

BSDi Internet Server (BSD/OS)

BSDi or BSD Inc. was founded in 1991 by some of the leading CSRG computer scientists. BSD/OS is a full-function, POSIX-compatible, Unix-like operating system for the 386, 486, and Pentium architectures. BSDI believes in one-stop shopping, high levels of integration and a product that requires payment of no external licensing fees.

FreeBSD

Developed and maintained by a large team of individuals. FreeBSD is a full function, POSIX-compatible, Unix-like operating system for Intel compatible (x86), DEC Alpha and PC-98 architectures.

NetBSD

Developed and maintained by a large team of individuals. NetBSD is another free version of BSD compatible with a very large variety of platforms, from 64-bit Alpha servers to handheld devices.

OpenBSD

Developed and maintained by a large team of individuals. OpenBSD is multi-platform 4.4BSD-based Unix-like operating system.

Relationship to Research Unix

Starting with the 8th Edition, versions of Research Unix at Bell Labs had a close relationship to BSD. This began when 4.1cBSD for the VAX was used as the basis for Research Unix 8th Edition. This continued in subsequent versions, such as the 9th Edition, which incorporated source code and improvements from 4.3BSD. The result was that these later versions of Research Unix were closer to BSD than they were to System V. In a Usenet posting from 2000, Dennis Ritchie described this relationship between BSD and Research Unix.

Relationship to System V

Eric S. Raymond summarizes the longstanding relationship between System V and BSD, stating, "The divide was roughly between longhairs and shorthairs; programmers and technical people tended to line up with Berkeley and BSD, more business-oriented types with AT&T and System V.

In 1989, David A. Curry wrote about the differences between BSD and System V. He characterized System V as being often regarded as the "standard Unix." However, he described BSD as more popular among university and government computer centers, due to its advanced features and performance.

Berkeley sockets

Berkeley's Unix was the first Unix to include libraries supporting the Internet Protocol stacks: Berkeley sockets. A Unix implementation of IP's predecessor, the ARPAnet's NCP, with FTP and Telnet clients, had been produced at U. Illinois in 1975, and was available at Berkeley. However, the memory scarcity on the PDP-11 forced a complicated design and performance problems.

Binary compatibility

BSD operating systems can run much native software of several other operating systems on the same architecture, using a binary compatibility layer. Much simpler and faster than emulation, this allows, for instance, applications intended for Linux to be run at effectively full speed. This makes BSDs not only suitable for server environments, but also for workstation ones, given the increasing availability of commercial or closed-source software for Linux only. This also allows administrators to migrate legacy commercial applications, which may have only supported commercial Unix variants, to a more modern operating system, retaining the functionality of such applications until they can be replaced by a better alternative.

Standards adherence

Current BSD operating system variants support many of the common IEEE, ANSI, ISO, and POSIX standards, while retaining most of the traditional BSD behavior. Like AT&T Unix, the BSD kernel is monolithic, meaning that device drivers in the kernel run in privileged mode, as part of the core of the operating system.

Significant BSD descendants Colored bar chart of BSD distributions usage

[OR]

b) Explain Free Software foundation and Specific Characteristics of Open Source Software Transformation

THE FREE SOFTWARE FOUNDATION

- "Free software is a matter of liberty, not price.
- To understand the concept, you should think of free as in **free speech (right)**, not as in **free beer** (gift).
- Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software.
  - The freedom to run the program, for any purpose (freedom 0).
  - The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
  - The freedom to redistribute copies so you can help your neighbor (freedom 2).

The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this."

"A program is free software if users have all of these freedoms. Thus, you should be 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.

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."

Very counter-culture

Hacker is considered a "good-guy"

"Hacker (computer security) someone involved in computer security/insecurity

Hacker (programmer subculture), a programmer subculture originating in the US academia in the 1960s, which is nowadays mainly notable for the free software/open source movement

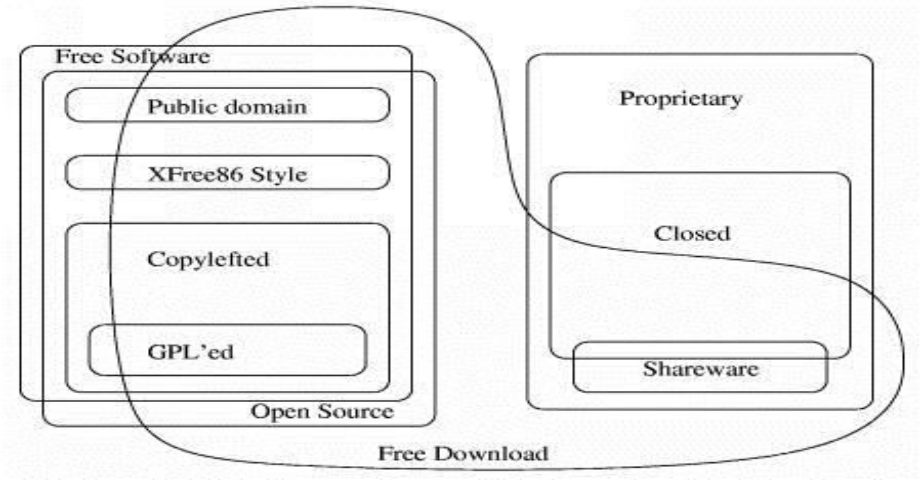
Hacker (hobbyist), an enthusiastic home computer hobbyist" http://en.wikipedia.org/wiki/Hacker

Cracker is a "bad-guy"

A cracker is someone who cracks software or digital media

"Software cracking is the modification of software to remove protection methods: copy protections, trial/demo version, serial number, hardware key, date checks, CD check or software annoyances like nag screens and adware".

- General Public License - GPL in 1991
  - The community rather than the company
  - Copyleft
  - No limits on software released under this license
- Opposite of proprietary software



SPECIFIC CHARACTERISTICS OF OPEN SOURCE SOFTWARE TRANSFORMATION

The development of open source software consists of planning, analysis, design, and implementation phases as in any other software model. However, there are unique characteristics of FOSS. In this section, we describe the main characteristic of Free and Open Source Software. In a typical FOSS, initially an individual or few volunteers involve in the project. Once the project is debut and successful then a community of project is established. Later other members from the community contribute to the project.

The Concurrent Versions System (CVS) helps is distributed development of FOSS. CVS is a client-server software revision control system. CVS keeps track of all changes in a set of files, and allows several developers to collaborate. CVS itself is a Free

and Open Source Software. Globally distributed software development by virtual teams promises the flexibility, responsiveness, lower costs, and improved resource utilization. Modular Design In modular design software architecture is divided into components called modules.

Modular design supports abstraction, increased understanding of the system and concurrent development. Due to distributed nature of FOSS, its design must be modular that can easily incorporate into the main system. Modularity is favorable characteristics for open source production. Modular design with well-defined interfaces helps in effective collaborative development of FOSS. Figure 1 shows the modular design approach of FOSS.

Reusability

Reusability means segment of source code that can be used again to add new functionalities with little or no modification. This fits very well the characteristics of the Open Source production process.FOSS licenses grants the rights to the developer to obtain the source code, inspect it, modify it, and distribute it. This mean FOSS licenses inherently encourages a developer to reuse

code. The reuse of code can be either within the project or outside the project, i.e., in other projects. A more details study with statistics of code reuse in open source software is conducted. FOSS repositories such as SourceForge offer huge amounts of reusable code.

Distribution and Licensing

Internet is the medium of distribution of Free and Open Source Software. Download websites, mailing-lists, blogs, forums, etc., all contribute to the wide spread publicity and distribution of Free and Open Source Software. Wide ranges of licensing options, such as GPL, LGPL, BSD,

ISC, Artistic License, etc., are available for FOSS distribution.

Reward Mechanisms

At the beginning of Free Software movement, seemingly it was difficult to perceive the business opportunities of Free and Open Source Software. But now business model of FOSS is getting success. Sources of income range from donations to providing services such as consulting, integration, support and training. It also worth to mention that reward other than money, such as reputation and serving community is also important for many developers.

23. a)Describe about Open Source Software Development Process.

THE OSS DEVELOPMENT PROCESS

Open-source software development is the process by which open-source software, or similar software whose source code is publicly available, is developed. These are software products available with its source code under an open-source license to study, change, and improve its design. Examples of some popular open-source software products are Mozilla Firefox, Google Chromium, Android, LibreOffice and the VLC media player. Open-source software development has been a large part of the creation of the World Wide Web as we know it, with Tim Berners-Lee contributing his HTML code development as the original platform upon which the internet is now built.

In his 1997 essay *The Cathedral and the Bazaar*, open-source evangelist Eric S. Raymond suggests a model for developing OSS known as the *bazaar* model. Raymond likens the development of software by traditional methodologies to building a cathedral, "carefully crafted by individual wizards or small bands of mages working in splendid isolation". He suggests that all software should be developed using the bazaar style, which he described as "a great babbling bazaar of differing agendas and approaches."

In the traditional model of development, which he called the *cathedral* model, development takes place in a centralized way. Roles are clearly defined. Roles include people dedicated to designing (the architects), people responsible for managing the project, and people responsible for implementation. Traditional software engineering follows the cathedral model.

The bazaar model, however, is different. In this model, roles are not clearly defined. Gregorio Robles suggests that software developed using the bazaar model should exhibit the following patterns:

Users should be treated as co-developers

The users are treated like co-developers and so they should have access to the source code of the software. Furthermore, users are encouraged to submit additions to the software, code fixes for the software, bug reports, documentation etc. Having more co-developers increases the rate at which the software evolves. Linus's law states, "Given enough eyeballs all bugs are shallow." This means that if many users view the source code, they will eventually find all bugs and suggest how to fix them. Note that some users have advanced programming skills, and furthermore, each user's machine provides an additional testing environment. This new testing environment offers that ability to find and fix a new bug.

Early releases

The first version of the software should be released as early as possible so as to increase one's chances of finding co-developers early.

Frequent integration

Code changes should be integrated (merged into a shared code base) as often as possible so as to avoid the overhead of fixing a large number of bugs at the end of the project life cycle. Some open source projects have nightly builds where integration is done automatically on a daily basis.

Several versions

There should be at least two versions of the software. There should be a buggier version with more features and a more stable version with fewer features. The buggy version (also called the development version) is for users who want the immediate use of the latest features, and are willing to accept the risk of using code that is not yet thoroughly tested. The users can then act as co-developers, reporting bugs and providing bug fixes.

High modularization

The general structure of the software should be modular allowing for parallel development on independent components.

Dynamic decision making structure

There is a need for a decision making structure, whether formal or informal, that makes strategic decisions depending on changing user requirements and other factors. Cf. Extreme programming.

Data suggests, however, that OSS is not quite as democratic as the bazaar model suggests. An analysis of five billion bytes of free/open source code by 31,999 developers shows that 74% of the code was written by the most active 10% of authors. The average number of authors involved in a project was 5.1, with the median at 2.

Open source software is usually easier to obtain than proprietary software, often resulting in increased use. Additionally, the availability of an open source implementation of a standard can increase adoption of that standard. It has also helped to build developer loyalty as developers feel empowered and have a sense of ownership of the end product.

Moreover, lower costs of marketing and logistical services are needed for OSS. OSS also helps companies keep abreast of technology developments. It is a good tool to promote a company's image, including its commercial products. The OSS development approach has helped produce reliable, high quality software quickly and inexpensively.

Open source development offers the potential for a more flexible technology and quicker innovation. It is said to be more reliable since it typically has thousands of independent programmers testing and fixing bugs of the software. Open source is not dependent on the company or author that originally created it. Even if the company fails, the code continues to exist and be developed by its users. Also, it uses open standards accessible to everyone; thus, it does not have the problem of incompatible formats that exist in proprietary software.

It is flexible because modular systems allow programmers to build custom interfaces, or add new abilities to it and it is innovative since open source programs are the product of collaboration among a large number of different programmers. The mix of divergent perspectives, corporate objectives, and personal goals speeds up innovation.

Moreover, free software can be developed in accord with purely technical requirements. It does not require thinking about commercial pressure that often degrades the quality of the software. Commercial pressures make traditional software developers pay more attention to customers' requirements than to security requirements, since such features are somewhat invisible to the customer.

It is sometimes said that the open source development process may not be well defined and the stages in the development process, such as system testing and documentation may be ignored. However this is only true for small (mostly single programmer) projects. Larger, successful projects do define and enforce at least some rules as they need them to make the teamwork possible. In the most complex projects these rules may be as strict as reviewing even minor change by two independent developers.

Not all OSS initiatives have been successful, for example SourceXchange and Eazel. Software experts and researchers who are not convinced by open source's ability to produce quality systems identify the unclear process, the late defect discovery and the lack of any empirical evidence as the most important problems (collected data concerning productivity and quality). It is also difficult to design a commercially sound business model around the open source paradigm. Consequently, only technical requirements may be satisfied and not the ones of the market. In terms of security, open source may allow hackers to know about the weaknesses or loopholes of the software more easily than closed-source software. It depends on control mechanisms in order to create effective performance of autonomous agents who participate in virtual organizations.

Development tools

In OSS development, tools are used to support the development of the product and the development process itself.

Revision control systems such as Concurrent Versions System (CVS) and later Subversion (SVN) and Git are examples of tools, often themselves open source, help manage the source code files and the changes to those files for a software project. The projects are frequently hosted and published on sites like Launchpad, Bitbucket, and GitHub.

Open source projects are often loosely organized with "little formalised process modelling or support", but utilities such as issue trackers are often used to organize open

source software development. Commonly used bugtrackers includeBugzilla and Redmine.

Tools such as mailing lists and IRC provide means of coordination among developers. Centralized code hosting sites also have social features that allow developers to communicate.

[OR]

b)Briefly describe about qualification and categorizing open source software.

QUALIFICATION AND CATEGORIZING: DEFINING OPEN SOURCE SOFTWARE

The Qualification and Selection of Open Source software (QSOS) is a methodology for assessing Free/Libre Open Source Software. This methodology is released under the GFDL license. Several methods have been created to define an assessment process for free/open-source software. Some focus on some aspects like the maturity, the durability and the strategy of the organization around the open-source project itself. Other methodologies add functional aspects to the assessment process.

Existing methodologies

There are more than 20 different OSS evaluation methods.

- Open Source Maturity Model (OSMM) from Capgemini
- Open Source Maturity Model (OSMM) from Navica
- Open Source Maturity Model (OSSMM) by Woods and Guliani
- Methodology of Qualification and Selection of Open Source software (QSOS)
- Open Business Readiness Rating (OpenBRR)
- Open Business Quality Rating (OpenBQR)
- QualiPSo
- QualiPSo Model for Open Source Software Trustworthiness (MOSST)
- Towards A Trustworthiness Model For Open Source Software: How to evaluate Open Source Software
- QualOSS – Quality of Open Source
- Evaluation Framework for Open Source Software
- A Quality Model for OSS Selection
- Atos Origin Method for Qualification and Selection of Open Source Software (QSOS)
- Observatory for Innovation and Technological transfer on Open Source software (OITOS)
- Framework for OS Critical Systems Evaluation (FOCSE)

General approach

QSOS defines 4 steps that are part of an iterative process:

- 1 - Define and organise what will be assessed (common Open Source criteria and risks and technical domain specific functionalities),
- 2 - Assess the competing software against the criteria defined above and score these criteria individually,
- 3 - Qualify your evaluation by organising criteria into evaluation axes, and defining filtering (weightings, etc.) related to your context,
- 4 - Select the appropriate OSS by scoring all competing software using the filtering system designed in step 3.

Output documents

This process generates software assessing sheets as well as comparison grids. These comparison grids eventually assist the user to choose the right software depending on the context. These documents are also released under the free GNU FDL License. This allows them to be reused and improved as well as to remain more objective. Assessment sheets are stored using an XML-based format.

Tools

Several tools distributed under the GPL license are provided to help users manipulate QSOS documents:

- Template editor: QSOS XUL Template Editor
- Assessment sheets editors:
  - QSOS XUL Editor
  - QSOS Qt Editor

- QSOS Java Editor (under development)