

Code Distribution Process - Definition of Evaluation Metrics



Project Acronym	EDOS
Project Full Title	Environment for the Development and Distribution of Open Source Software
Project #	FP6-IST-004312
Contact Author	Radu POP, rpop@mandrakesoft.com
Authors List	Ciaran Bryce, Michel Pawlak, Michel Deriaz - Universite de Geneve Serge Abiteboul, Boris Vrdoljak - INRIA Gemo Project Tova Milo, Assaf Sagi - Tel-Aviv University Stephane Lauriere, Florent Villard, Radu POP - MandrakeSoft
Workpackage #	WP 4
Deliverable #	1
Document Type	Report
Version	1.0
Date	March 22, 2005
Distribution	Consortium, Commission and Reviewers.

Chapter 1

Introduction

This document proposes a measurement and evaluation methodology and defines the metrics that we consider as important in EDOS for the code distribution process for Free and Open Source Software (F/OSS). Our aim in attempting to define the metrics is the following:

- Clarify our understanding of the F/OSS code distribution process, and subsequently to help identify areas for improvement.
- Enumerate what needs to be measured in the F/OSS code distribution process. The purpose of measurement and evaluation is to compare different architectures for code distribution the existing one and those that will be proposed in the EDOS project.

Measurement and evaluation have been facets of software engineering for some time. ISO (the International Organisation for Standardisation) and IEC (the International Electrotechnical Commission) have established a joint technical committee for worldwide standardization in the field of information technology. They have developed a set of standards for software product quality relating to the definition of quality models (the 9126 series) and to the evaluation process (the 14598 series). A quality model defines the characteristics of a system to be measured and the metrics that evaluate how the system to be measured performs with respect to these characteristics.

The ISO/IEC 9126 standard defines characteristics that relate to software quality, i.e., functionality, reliability, usability, maintainability and portability. However, EDOS certainly requires a particular quality model since the characteristics that we wish to measure relate to a service that implements

the code distribution process and its operation, rather than just the software itself. ISO/IEC 9126 nonetheless gives useful advice on the definition of metrics. For instance, it argues that the metrics should allow for a reliable measuring process, yield a coherent correlation to human judgement, and be automatable if possible, etc.

The ISO/IEC 14598 standard series is concerned with the process of evaluation of quality models seen from different viewpoints, e.g., developers, acquirers, and evaluators. This is a useful guideline since the F/OSS code distribution process should also be measured from the viewpoint of different actors, i.e., the editor, developers, end-users, etc.

In the measurement and evaluation process, we also take into consideration some elements of the Goal/Question/Metric method, which was developed at NASA and mostly used for the improvement of software development. The method leans on the principle that each measurement must be focused on specific explicitly stated goal in order to be effective. A set of questions is derived from the goals to define what it means to satisfy those goals as completely as possible in a quantifiable way. There can be several layers of questions, which resembles the layers of quality model as defined in ISO 9126. Finally, a significant and minimal set of metrics is identified in order to answer the questions adequately. As a result, all the metrics are related to the goals of the measurement activity.

Chapter 2 presents a survey on the code distribution process in the most well known F/OSS projects. The main Linux distributions are described, as well as file sharing, bug-tracking and ticketing systems. Special attention is given to Mandrakelinux distribution, which represents the focus of interest of our measurement and evaluation process.

Related work for EDOS Project includes the approaches taken by other Linux editors and by other free software distributors such as the BSD operating systems, by software development systems like Apache or Mozilla, as well as peer-to-peer based file sharing systems like BitTorrent or Kazaa. Of interest are the mirroring techniques used but also process management, e.g., how testing and QA are organised by the editors.

Of all the related work, the RedHat Distribution Network [15] may in fact be the most pertinent, since it's architecture is defined at a functional level and includes core abstractions necessary for the code distribution process, and then refined to an implementation architecture.

Chapter 3 defines the characteristics of the F/OSS code distribution quality model, where each characteristic can have sub-characteristics. Each terminal

node in this hierarchical structure of characteristics should have at least one metric associated with it. The metrics highlight areas where the process can be improved by EDOS project and serve as a reference framework for comparing different architectures of the code distribution process.

Chapter 2

Survey on the Existing Systems

2.1 Operating Systems

This subsection briefly looks at the main Linux distributions and BSDs as these are the current alternatives to Mandrakelinux.

A Linux distribution or GNU/Linux distribution (or a distro) is a Unix-like operating system plus application software comprising the Linux kernel, the GNU operating system, assorted free software and sometimes proprietary software, all created by individuals, groups or organizations from around the world.

Companies such as Red Hat, SuSE and MandrakeSoft, as well as community projects such as Debian, assemble and test the software and provide it as a complete system, more or less ready to install and use. There are over 200 different Linux distributions in active development.

Following is a diagram sketching the basic interactions in today's F/OSS processes, from upstream development to installation on the end user's machine.

2.1.1 Red Hat

Red Hat Linux split into two directions in 2003. One branch merged with Fedora, and is also known as the Red Hat community edition. The second became the commercial Red Hat Enterprise edition. The key legacy of Red Hat is its packaging technology – RPM – that is used today by several F/OSS projects.

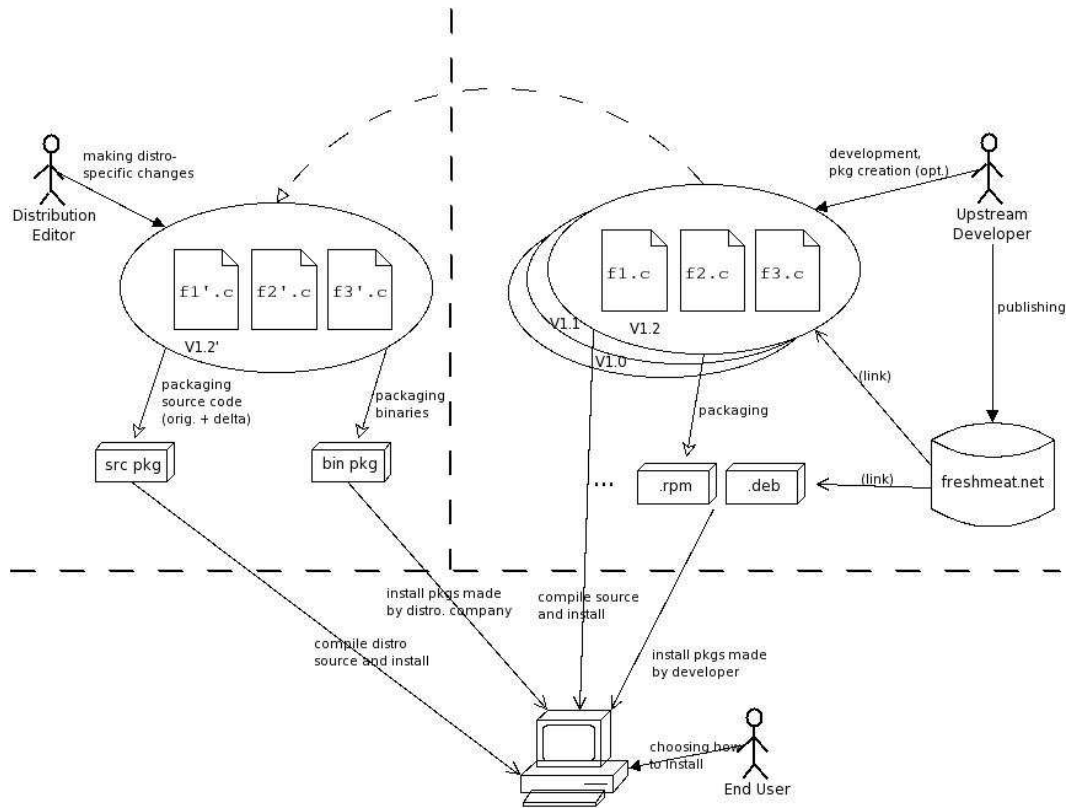


Figure 2.1: Basic Interactions in F/OSS Processes

Red Hat Network

Binaries for Red Hat Enterprise Linux are no longer provided via ftp across mirrors, but rather by a customised management architectural solution called *Red Hat Network* (RHN). Customers use RHN to download distribution ISO's, errata (patches) and software packages. Clients that subscribe to RHN can automatically update their system in a customised way.

Two architectural models exist for RHN. The first is the *Hosted Model* where the distribution is stored on the network. This model is recommended for individuals or small companies. The second model is the *Satellite Model* and is used by bigger enterprises. It consists of placing an RHN on the customer's local network. A satellite server then serves the different client machines and connects to Red Hat in order to download updates. In this way, each client machine in the local network can use a different Linux configuration.

All communication between customers, managed systems and RHN is protected by SSL encryption for privacy and authentication. Every package (RPM) is gpg-signed and contains MD5 checksums for both the package and contained files to ensure data integrity before deploying on target systems.

An interesting aspect of the RHN is that a functional definition is presented in [15]. The network is accessible through an *Access API*. The key abstraction is the *channel*, which corresponds to a set of packages and every client machine that is connected to a specific channel can be updated when the content of the channel changes. Channels can be created and managed by the system administrator. One possibility is for him to associate access rights with a channel and thus control the local systems that read from it.

A channel can be used to implement a *staged environment*. Along with the base channel that corresponds to the core system, other types of channels exist. A development channel is used by developers of the community to distribute their work. A Testing & QA channel is used to report on the packages under development and for bug reports. A production channel is used to develop beta versions. The architecture allows actions to be defined on channels by users. An example action could be to remove packages whenever a new version is available, or to rollback to a previous version of the system when a compilation error occurs. Another use case is a system administrator that downloads new patches and tests them on specific machines. If the test passes, he copies the updates in the production channel, where the users' machines are connected.

The Red Hat Network has two useful lessons for us.

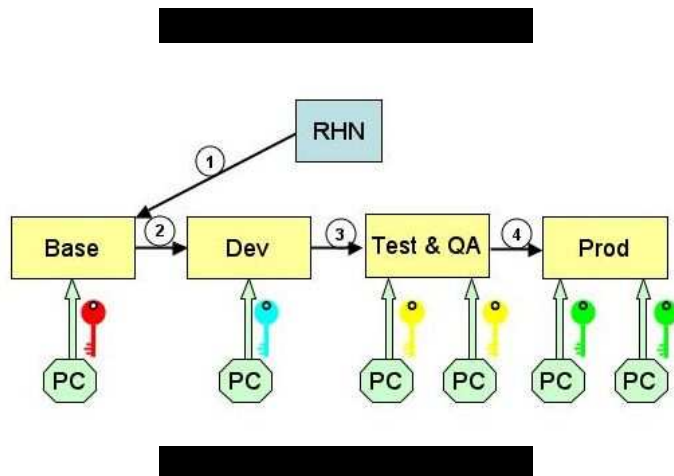


Figure 2.2: The Red Hat Network's Staged Architecture

1. The network suggests that distributing software to end-users is not independent of other F/OSS aspects. RHN captures a large slice of the *open source process* since it deals with installing, testing, QA and feedback.
2. A functional architecture is defined that captures all major requirements of the distribution network. This can then be refined to specific architectures. This approach allows one to consider the requirements of the system independently of the underlying platform. Such an approach could be very promising for EDOS.

Fedora Project

The goal of the Fedora project is to work with the Linux community to build a complete, general purpose operating system exclusively from free software. A stable release is usually provided 2 or 3 times per year, and selected components from the distribution are chosen for incorporation into RedHat Enterprise Linux. Fedora is distributed through mirror servers¹ (there were 222 mirrors in operation on the 26th of October 2004) and also via Torrent. The developer community is quite proactive, and bug reports are maintained via a Bugzilla site.

¹<http://fedora.redhat.com/download/mirrors.html>

2.1.2 Debian

Debian is community project whose aim is to provide a free operating system based on the Linux kernel. The project organisation is funded by donations from industry. The development community reportedly is composed of thousands of developers.

The Debian Linux distribution network is based on mirror servers. All mirrors seem to be maintained by owners who need not be part of the Debian project.

Debian has 32 official mirrors² (one in each major client country) and about 340 non-official ones. The main difference is that an official mirror (with a name like `country.debian.org/debian`) must be updated at least once a day and support *push mirroring*. This is a technique that allows a server to inform and update its client mirrors as soon as it receives a new version. Mirrors are therefore hierarchically organized in two levels.

The time taken to effect a copy for pull-mirroring can vary, so each mirror contains a timestamp, accessible at <http://mirror.debian.org/status.html>. Analysis of this log reveals that several mirror servers are not well maintained. A client (leaf) mirror simply compares its own time-stamp with its server mirror at pre-configured time intervals.

The size of a Debian release is about 8 GB for a supported architecture, and the whole thing is 100 GB. A distribution is composed of 8710 packages. A mirror contains a U.S. distribution version and a non-U.S. distribution version to avoid legal problems arising from U.S. patent law and encryption export restrictions.

The Debian distribution process, and the problems posed, is quite similar to that of Mandrakelinux, e.g., short release cycles and poorly maintained mirrors. The main difference seems to be the push mirroring used by primary mirrors that Debian employs.

Debian has three distributions: "Stable", "Testing" and "Unstable". Actually there are two more distributions - "Experimental", which contains volatile elements which - should they have bugs - may bring down the whole system (for example, a new file system), and "Frozen" which is a temporary distribution before "Testing" becomes "Stable". The "Experimental" distribution is not meant for personal use, but rather as a platform for trying out new ideas and testing them. The first 3 distributions are considered okay for home use (even "Unstable", though not recommended for beginners).

²<http://www.debian.org/mirrors>

A new package usually gets into the "Unstable" distribution (though there are some exceptions, as noted here). This distribution contains packages which are supposed to be - on the whole - stable, according to their developers and sites like Freshmeat. However, those packages haven't been tested and integrated into the whole Debian distribution and so are considered for now to be "Unstable". It is important to note that because the packages in "Unstable" do have some degree of stability, there are some users who prefer to have the "Unstable" distribution installed on their machines - just to be among the first to get new and updated software.

An automatic process evaluates nightly the packages in the "Unstable" distribution. If certain criteria for a package are met (spent X days in "Unstable", has fewer critical bugs than its respective version on "Testing" and additional criteria) then the package is moved to "Testing". "Testing" is the distribution which is the release candidate.

Whenever the Debian release manager decides (which is not very often) a freeze is declared on the "Testing" distribution. At that point buggy packages are removed from the distribution and no new packages can be let in except for bug fixes. After an additional period of time the distribution goes into a "deep freeze" when no changes at all are allowed, except installation-related. When the distribution proves to be stable enough - it becomes the new "Stable" distribution and distributed as such. As implied before, the "Stable" distribution is not updated very often and so fits corporate users and servers, where keeping up with the "bleeding edge" is not a requirement.

Debian contributors make changes to packages' source code for them to fit with the whole Debian distribution, and the changes are kept alongside the original source. However, the Debian hierarchy has no "internal" and "external" distinction among contributors. Practically every one can become a maintainer of one package or more. The maintainer is actually the one responsible for uploading packages to the various distributions, while the developers send their source code and diff files to the maintainers.

2.1.3 FreeBSD

Berkeley Software Distribution, or BSD for short, refers to a set of versions of the Unix operating system. The three principal free variants of BSD are FreeBSD, OpenBSD and NetBSD. This section describes the approach used by the FreeBSD release engineering team to make production quality releases of the FreeBSD Operating System [12] as well as the FreeBSD approach to making available and installing applications [1].

Development process

The development of FreeBSD is a very open process. FreeBSD is comprised of contributions from thousands of people around the world. Although the FreeBSD Project provides anonymous CVS allowing the community to review and contribute to the code, only a group of around 300 people are given write access to the CVS repository. These people are called *committers* and are responsible for the bulk of FreeBSD development. An elected core-team of very senior developers is responsible for deciding the project's overall goals and directions.

In order to facilitate the rapid development of production quality releases, FreeBSD development has been split into two parallel tracks. The main development branch is the *HEAD* of the CVS tree, known as "FreeBSD-CURRENT" or "-CURRENT". This branch is the "bleeding-edge" of FreeBSD development through which all new changes first enter the system. A more stable branch aimed at production environments, known as "FreeBSD-STABLE" or "-STABLE", is also maintained. Changes go from -CURRENT to -STABLE at a different pace, and with the general assumption that they have been thoroughly tested by the user community. This approach allows FreeBSD to provide a high security environment while continuing to improve the system and implementing new technologies and features. Both branches are located on a master CVS repository and are replicated via CVSup to mirrors all over the world.

Bug reports and feature requests are continuously submitted by users throughout the release cycle. Problem reports are entered into FreeBSD GNATS [7] database through email, the *send-pr* application, or via a web interface.

Release process

The FreeBSD Release Process is based on a standardized release engineering procedure. This procedure emphasises the security and stability of FreeBSD releases and refuses to sacrifice these features for any self-imposed deadlines or target release dates.

New releases of FreeBSD are released from the -STABLE branch at approximately four month intervals. 45 days before the anticipated release date, the release engineer sends an email to the development mailing lists to remind developers that they only have 15 days to integrate new changes before the code freeze. This process is known as "MFC sweeps" ("Merge From CURRENT") and it describes the process of merging a tested change from

the -CURRENT development branch to the -STABLE branch. Once the code enters the “Code freeze” state, it becomes much harder to justify new changes to the system unless a serious bug-fix or security issue is involved. Then, until final release is ready, at least one release candidate is released per week, the release engineering team being in constant communication with the security-officer team, documentation and port maintainers. When several candidates have been made available and all major issues have been resolved, a new branch is created for the release, the version number is bumped up and Release Tags are created. Only then is the new Release officially created.

For most conservative users, individual release branches were introduced with FreeBSD 4.3. These release branches are created shortly before a final release is made. After the release goes out, only the most critical security fixes and additions are merged onto the release branch.

Distribution process

FreeBSD is available from anonymous FTP sites and from CDROM.

The official FreeBSD public FTP sites are all mirrors of a master server that is open only to other FTP sites. When the release has been thoroughly tested and packaged for distribution, the master FTP site is updated. It may then take between several hours and two days before a majority of the Tier-1 FTP sites have the new software. Release engineers coordinate with the FreeBSD mirror site administrators before announcing the general availability of new software on the FTP sites. FreeBSD’s handbook advises mirrors to load the release package set at least four days prior to release day. Thus the release is uploaded between 24 and 48 hours before the planned release time with “other” file permissions turned off. This allows mirror sites to prepare availability of new releases while avoiding that users start downloading it from mirror sites.

During the period between releases, nightly snapshots are built automatically by the FreeBSD Project build machines. The user community can keep their system up to date with -STABLE and -CURRENT development using CVSup and “make world” tools in order to download and apply latest patch sets to their system source code tree.

CVSup can mirror different kind of files like sources, binaries or symbolic links. It parses and understands the Revision Control System (RCS) files of a CVS repository, and continually keeps track of updates made on files. Performance is obtained through the use of a multi-threaded architecture on both client and server, which allows for more efficient use of both the upload

and download channels. The authors claim that it is the fastest mirroring process available since it uses better the available bandwidth of the network. While in traditional systems the server sends a list of its files to clients, and then sends the files that need to be updated, a CVSup client creates a list of its files, sends the list to the server, and waits for the file updates.

Ports Collection and Packages

The FreeBSD *ports collection* is the main system for installing new software versions on machines running FreeBSD. The FreeBSD web site maintains an up-to-date searchable list of all available *ported* applications.

A FreeBSD port for an application is a collection of files designed to automate the process of installing an application from source code, i.e. downloading needed files, applying patches, installing dependencies, compiling the application then installing it. Amongst other advantages, unlike packages, ports allow users to compile applications with tweaked, non conservative, options specific to their environment. They also allow users to use application-specific compile time options and allow them to apply latest existing patches. Note that binary packages for most important ports are also available from FreeBSD servers, and that packages can be generated from ports tree.

As for system source code tree, FreeBSD port tree can be updated and kept up-to-date using CVSup. Once the port tree has been updated, installed ports can be updated using the *portupgrade* tool. Ports security check is ensured by the *portaudit* tool which checks FreeBSD database for known ports issues. Once installed *portaudit* is automatically run at ports installation time and can be run on a regular basis to check already installed ports.

In FreeBSD, anyone may submit a new port, or volunteer to maintain an existing port if it is unmaintained, not needing any special commit privileges. The guidelines for creating and maintaining ports can be found in the *Porter's Handbook* [2].

2.1.4 Mandrakelinux

Mandrakelinux is a Linux distribution created by MandrakeSoft. The first release was based on Red Hat Linux (version 5.1) and KDE (version 1.0) in July 1998. It has since diverged from Red Hat and has included a number of original tools mostly to ease system configuration.

MandrakeSoft's development version of the next Mandrakelinux release is

called "Cooker". The purpose of Cooker is to improve the Mandrakelinux distribution by permitting a better interaction between the development team and the Mandrakelinux users, both for debugging and adding new features. It is an entire distribution unto itself, that is constantly in progress and sometimes cannot even be installed because it is broken itself due to the incompatibilities.

Cooker is by all means a distribution, albeit it might be unstable because it is in testing status. About every 6 months a new stable release is out. Before the release (about 3 months before) a beta version is already out for users to play around with and submit bugs. Later, as testing (and subsequent fixing) proceeds, the version becomes more stable and is declared a release candidate (RC).

The packages in a Mandrakelinux distribution are divided into two categories: *main* and *contrib*. *Main* includes the packages which are essentially the "sponsored" release. These packages have been tested and verified before making it into the next release. As the Cooker version becomes more and more stable, "freezes" are declared and no more new contributions to the packages are permitted, except for fixing serious bugs.

The other category, *contrib*, contains pieces of software which are not part of the core of the distribution, but they are still supposed to work along with the release. When "freezes" are declared, it is still possible to contribute and submit new and updated packages to contrib.

Whenever a contributor packages a new piece of software, she has to put it in the "incoming" folder of the MandrakeSoft's FTP server. Also, she has to notify the Cooker mailing list and the editor, so he will know that it exists and that he has to decide what to do with it. Contributors are encouraged to package only the source code (source packages) and not the binaries, since the editor has to perform some validation tests on the code (to prevent trojan horses, non-licensed software, other legal problems and so on).

Source code is often changed to fit with Mandrakelinux distribution. Usually, the contributor who packaged the software also makes some changes. In other cases, it will be the editor's duty to tweak the code. Either way, the original sources are kept along with a diff file containing all the changes that were made.

Each package in the distribution has a *maintainer*. The *maintainers* are persons who are "trusted" by MandrakeSoft. A new contributor is called "external contributor" and he can only upload packages in the way described previously. He can not be their maintainer. The maintainer is an "internal

contributor”. Those are people who were once ”external contributors” but were considered trustworthy by the editor due to their activities up until now.

The Mandrakelinux distribution process can be described by the figure below:

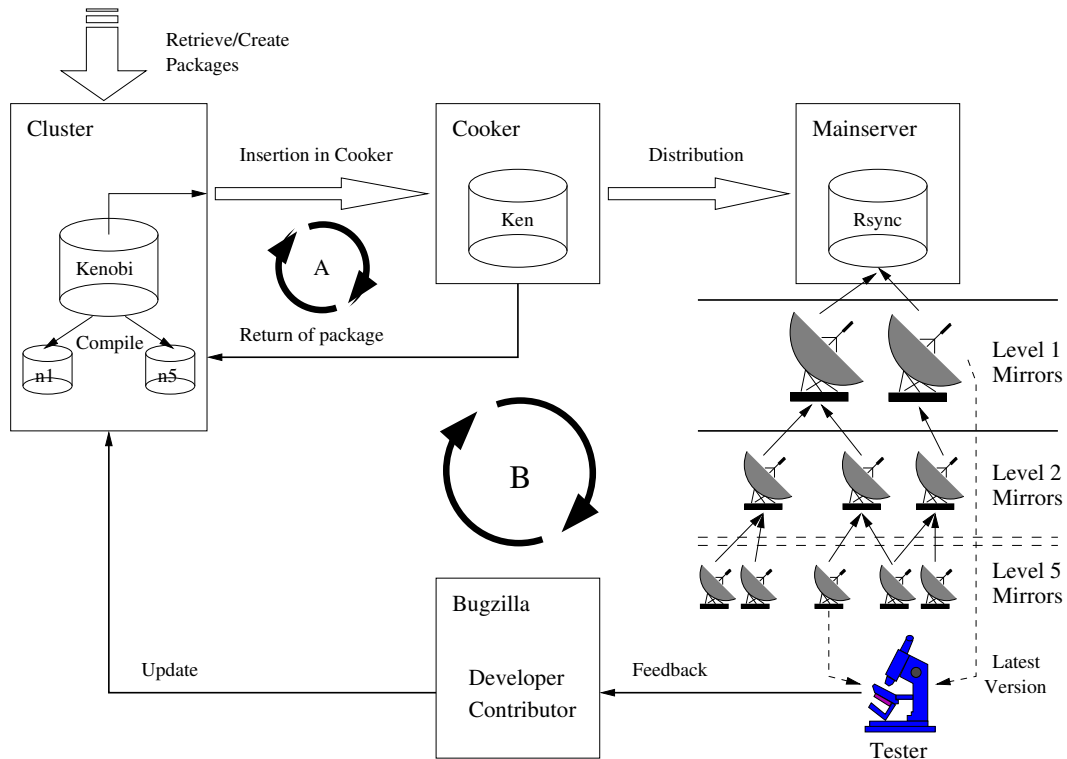


Figure 2.3: The Mandrakelinux Distribution Process

As depicted also in the figure, in Mandrakelinux developing process we can identify two main cycles performed in the preparing process for a new release.

The first one, marked with A, is rather an internal cycle, specific to MandrakeSoft’s package maintainers. Each maintainer is a Mandrakelinux developer in charge of a particular set of packages, who searches for the last version of package, builds it on the machines in the cluster and inserts it into Cooker. At this point, when a new version of a package is uploaded into Cooker (e.g.: a new version of Perl library), some inconsistencies may appear between the new package and the dependency related packages (e.g.: applications using the Perl library). Therefore, the maintainer must check the packages affected by the last update and return them to the cluster. They are rebuilt here and reinserted afterwards in Cooker.

The number of people implied in this development cycle is restrained and limited only to the package maintainers. The regular developers and contributors are not allowed to add or to modify packages in Cooker.

On the other hand, the second cycle is much more larger and involves a lot more people. It represents the way of taking benefit from the Mandrakelinux community's contribution. As we mentioned before, the role of Cooker is to provide the community with the last versions of packages included in the Mandrakelinux release currently under development. The distribution process is done in the classic fashion, using mirror sites organised in a multi-level hierarchy.

The first step of the distribution consists in replicating the whole Cooker release on a Mainserver using *rsync* for synchronization. MandrakeSoft disposes of a fixed set of primary mirror servers, called also "level one mirrors", which hold copies of the Mainserver. The primary mirrors get the updates using either push or pull method. Each primary mirror replicates the whole content of the Cooker release, meaning both source and binary packages, main and contributors packages, for all architectures.

The secondary mirrors get synchronized afterwards with the level one mirrors. In Mandrakelinux distribution the hierarchy of mirror servers goes up to 5 levels, but the autonomy of the secondary mirrors is rather strong. Therefore a strict control on the content of each secondary mirror or on the mirrors' network architecture can not be achieved. Each mirror decides for its own to which mirror to synchronize, on which time interval, and what content to replicate. Secondary mirrors use the pull method for synchronization.

Using one of the mirror servers, the user is able to receive the last version of the packages she is interested in. It is about a particular category of users, the ones that are willing to tryout and to test the latest features and improvements of the applications.

Users' feedback is done by Cooker's mailing list and Bugzilla reports.

2.1.5 Other Linux Distributions

SuSE

SuSE is a major retail Linux distribution, produced in Germany and it's now currently owned by Novell.Inc.

SUSE Linux was originally based on Slackware Linux and it was founded in late 1992 as a UNIX consulting group, which among other things regularly

released software packages that included SLS and Slackware, and printed UNIX/Linux manuals. They released the first CD version of SLS/Slackware in 1994, under the name S.u.S.E. Linux1.0. The name "S.u.S.E.", later shortened to just "SuSE", was originally an acronym for the German phrase "Software und System Entwicklung" ("Software and system development"). Unlike most other makers of Linux distributions who allow immediate download of their final versions, SUSE first releases the Personal and Professional versions in boxed sets which include extensive documentation, then waits a few months before it releases versions on its FTP servers.

Gentoo

Gentoo Linux is another popular Linux distribution.

Even if its creator and former software architect, Daniel Robbins, imported the "Ports" system from the FreeBSD community, he constructed the distribution around a specific philosophy. First he wants Gentoo to remain free. Secondly, he wants that users maintain complete control over their machines. This last point is important since it differs from the way a distribution like Mandrake Linux works. Mandrake Linux furnishes software that is responsible for installing, uninstalling or updating packages. This works transparently and is comparable to Windows systems. On the other hand, Portage (the "Ports" system of Gentoo) uses scripts to describe which, when and how packages are updated. The user configures his system exactly the way he wants. Even if a particular system evolves automatically over time (depending on how the user configured Portage), Gentoo also provides some "official" releases on CD-ROMs, through mirrors servers or via BitTorrent. Gentoo CVS servers can also be accessed over the Web.

Slackware

There is not a lot of documentation on the Slackware website. Their philosophy claims that they want to be the most "Unix-like" Linux distribution. Graphically we would represent Slackware as the intersection of Debian, Gentoo and LFS (Linux From Scratch). Slackware can be obtained through CDs, via BitTorrent, or via a mirror server.

Knoppix

Knoppix is a bootable CD-ROM containing a full Debian-based Linux distribution. No installation is required. The Knoppix CD automatically recognizes the hardware, launches a Linux kernel and then unzips and launches the different applications following user requests. An ISO image of this CD can be freely downloaded from the Knoppix website.

2.2 Projects

This section looks at other – non-Unix – F/OSS projects that have some lessons on code distribution.

2.2.1 Apache

Apache is a software foundation promoting the development of free and high quality software. Developers are volunteers who communicate only via mailing lists in order to keep a trace of the contents and to allow people to work in an asynchronous manner. This last point is essential since the developers are dispersed over the world and often work on the project during their spare time.

Politically, Apache does not employ a hierarchical structure to co-ordinate projects. They opt for a *meritocracy* – the more you contribute, the more power you get. Anybody can take part in any of the Apache projects. A newbie typically starts by participating in a mailing list, contributing later by sending patches, and little by little, he becomes trusted by the other community members. He can then be granted direct access to the source code.

When decisions need to be taken, the community uses a basic voting system. The mailing list publishes the topic of the vote and a deadline, typically 72 hours. To vote, community members answer with "-1", "0", or "1" if they respectively disagree, have no opinion, or agree. Depending the case, a "-1" vote can be interpreted as a veto. In this case the vote is frozen until an agreement is found and all the members withdraw their negative vote.

The Apache Software Foundation (ASF) supervises the different projects through its Board of Directors. The board essentially deals with with political issues. All technical issues are delegated to each Project Management Committee. Despite this liberty, the different projects are organized in sim-

ilar ways. Sources are stored in CVS servers that can be updated several times a day. Regression tests are provided with the sources. A developer that improves a code module then applies all available regression tests before asking its machine to automatically produce the patch (via CVS). This patch is then published, and the regression tests are updated.

We can already guess some strong relations with the EDOS project. Nonetheless, each of Apache's projects are independent of each other, and that the average size of one is significantly smaller than a Linux distribution.

2.2.2 Mozilla

Mozilla functions in a very similar way to the Apache Foundation. They also use the meritocracy as a political pillar and the same tools to coordinate development (CVS, Bugzilla...). They work on 6 different projects: Firefox, Thunderbird, Mozilla Suite, Bugzilla, Camino and Calendar Project. The Mozilla Foundation, created in July 2003, deals with organizational, legal, and financial issues for the Mozilla open-source software project. There are currently five members in the Mozilla Foundation Board of Directors. Mozilla.org is the central point that will maintain mailing lists, provide technical and architectural direction for the projects, collect changes and make periodically releases. New code is however essentially developed among the community members, of which there are currently several thousand. A patch or any modification from a community member is sent to the owner of the corresponding module (mozilla.org designs the different module owners), who includes it after testing.

One difference between Apache and Mozilla is that Mozilla does not use a voting system in order to take decisions. The Mozilla model is based on commercial software development processes. It is the module owner who decides what code gets included in his module and it is mozilla.org which decides which modules get introduced into the repository. The aim is to avoid several parallel versions of the software. Mozilla calls this the *Benevolent Dictator* system, because the Dictator (module owner or mozilla.org) has always to make the best choices for the community if he wants to keep his place. Since it is an open-source project, if the module owner does not do his job well, the community members just have to design a new module. This is also true for mozilla.org; if they do not meet the expectations of the module owners or the community members, another code assembler is designated.

An interesting site to mention here is mzodev.org, which contains currently 200 applications. The projects hosted on here create applications and add-

ons that are based on top of the source code provided by mozilla.org.

2.2.3 Open Office.org

OpenOffice has gained considerable success over the past few years as an alternative though compatible environment to MicroSoft Office. It runs on all major OS platforms, including Linux, MacOS and Windows. The project's APIs are open and use the XML standard for document representation.

OpenOffice is an off-shoot of StarOffice - a product bought by Sun Microsystems in 1999. The code base is written in C++, though APIs exist for other languages, including Java. The project is managed by a Community Council, one of whose goals is to oversee the status of the projects in progress. Projects can be classed as accepted, native language or incubator, and each has a designated lead assigned by the Council. The Council is supported by donations from the public. The software licenses used for OpenOffice distributions are LGPL and SISSL.

Software is distributed via a mirroring system. A two-tier mirroring set-up is employed with rsync being used to effect copies between them. A mirror is generally required to maintain two stable releases, and optionally a localised (to a country) release, a developer release and a contribution release (on which no QA has been effected yet).

In order to support the code distribution process, OpenOffice solicits different kinds of support from the community. The community can contribute documentation support - especially with respect to the different natural languages. Code contributions are made in response to issues posted by the project lead, and submissions are made via CVS.

The community is also involved in testing and quality assurance, and Issue Tracker - a follow-up to IssueZilla - is used to coordinate this. Users can contribute remarks, smoke tests - which are Web-based query forms - and can also run automated program unit tests (known as qadevOOo) that are written in Java.

2.2.4 Eclipse

Eclipse is a popular development environment used today that integrates several important development tools and has support for different languages. Its plug-in based architecture makes it extensible and it has now been deployed on a wide range of platforms. The environment is managed by the Eclipse

Foundation, which is a non-profit consortium of industry leaders, including Borland, Hitachi and Sybase.

Eclipse is organized as a series of projects and sub-projects, and each has a designated lead who is responsible for overseeing the project: ensuring that development subscribes to open source principles such as meritocracy and transparency. Leads must adhere to a set of process guidelines that are formalized in a document known as a charter.

Eclipse projects are distributed using a mirroring architecture. The distribution size for all projects combined is around 65 Gigabytes and nightly builds can be as large as 1 Gigabyte. There are around 100 mirrors currently in operation, each is independently maintained and uses an rsync script to effect copies. Mirror sites are requested to make a copy at least once per day.

Developers use CVS to contribute code to builds to a project. Bugzilla is used for bug tracking and reporting, along with the standard newsgroups and mailing lists.

2.3 FileSharing

This section looks at some file sharing systems. Our motivation is not because they are F/OSS projects, but because they are – and can be – used as the basis for a distribution architecture.

2.3.1 BitTorrent

BitTorrent allows users to download a file in a near peer-to-peer fashion. Instead of each user downloading from a centralised server. A user downloads different pieces of a file from different users. Thus, users download and upload simultaneously, and bandwidth is distributed between users. BitTorrent is used already by Mandrake Linux developers.

An interesting presentation of the resource consumption aspects of the system is presented in [4]. The system aims for Pareto efficiency (a system where resources are allocated in such a manner that no individual is better off or worse off), a higher level of resource utilization and robustness. The main problems that the system has to address are high churn rate, fairness, finding the best piece allocation strategy and ensuring steady up-rates. A specific problem is that users tend to kill their clients as soon as download completes (irrespective of on-going uploads). Peers use a tracker site to find each other

and it stores a minimum of information. In general the algorithm used by the tracker is to generate a random list of peers since this is the most robust with respect to disconnection and segmentation, resulting from churn. A tracker also stores a hash of each piece so that its integrity on receipt can be verified. A seed (complete version of the file) must exist and be downloadable in totality from there. The piece that a peer chooses for download can follow a strict priority, rarest first (i.e., the piece is the least common among the set of peers), random order, etc. Choking is the explicit (temporary) refusal to upload and is required for good system performance (i.e., it can be used to prevent imbalance in rates between two users) and is how Pareto efficiency is achieved.

2.3.2 Kazaa

Another system we investigated is file sharing via peer-to-peer networks. We choose Kazaa because it is a very well-known used system (even if not anymore the most popular) and because we found more documentation about this system than on others. It is clear that big differences exist between code distribution and file sharing, but we still find important to analyze more deeply P2P networks, and ideas can be reused in our project. We will not give a description here about how Kazaa works. Rather, we point out some particularities that can potentially be exploited in EDOS.

First we discovered that P2P is more and more used and that it consists today in the majority of internet traffic. Then we learned that P2P downloads does not follow the traditional Zipf's law, used for Web traffic. The curve is much flatter, giving less importance to popular files than predicted by the Zipf's law. This difference is explained by the fact that the same internet site is visited several times by the same user, while a file is usually downloaded only once by a particular user. In contrast to Web pages that evolve with the time, a shared file is always the same. And finally we learned also that Kazaa favors good peers. A peer that shares lots of files will obtain a better priority for its owns downloads. A good description of the system has been presented in [8] at ACM SOSP in 2003.

2.3.3 Other Systems: Google File System (GFS)

GFS³ [6] is used for all of the data processing requirements of Google. As with mass storage systems, the requirements include performance, availability,

³Google File System

reliability and scalability. It is also built from observations real usage. First, component failures are the norm and not the exception. Second, files are huge by traditional standards; multi Gigabytes are common and this influences block sizes. Third, files are modified nearly exclusively in append-only mode and this permits a relaxed consistency model. Fourth, the API is flexible to support further development; it supports record append and snapshot commands.

The architecture is composed of a master and several chunk servers. A chunk server stores file chunks (as local Linux files). The master maps file names and offsets to chunks, and stores all meta-data. Chunks are replicated on chunk servers.

Meta-data is optimized for recovery. For instance, chunk servers store information about chunks they have and the master queries these chunk servers when it boots. Only the operation log needs to be permanently stored; this keeps a log of changes to the meta data.

2.4 BugTracking

A large number of different *Bug-tracking and Ticketing tools* are available today. Each presents its own advantages and features. The main goals of these systems are to provide a database for bugs, to keep track of to-do lists as well as to prioritise, schedule and track dependencies. They define roles and responsibilities (e.g. "programmer", "integrator", "tester", ...) and specify who is working on what bug. This allows work duplication to be avoided and people can help out and provide feedback. Developers benefit by having an organized system for getting input from users and having a large pool of feedback for quality assurance. Users are allowed to submit bugs found in software directly to developers while also tracking the status of the work on those bugs

Bugzilla [3], a project of Mozilla, is one of the best known bug-tracking systems. It is web based, implemented in perl with MySQL as a back end, its solid in appearance and is used by a number of high-traffic web sites. Bugs that Bugzilla is tracking can be issues as well as requests for enhancement. Amongst other features, Bugzilla provides the ability to define to which component a bug is related, a status whiteboard used for writing short notes about the bug, keywords, targeted milestone estimating the earliest milestone at which a bug might be resolved and bug dependencies. Another feature of Bugzilla is to provide the ability to add attachments.

RedHat Bugzilla [9] is a variant of Bugzilla that can work with Oracle, MySQL, and PostgreSQL databases serving as the back-end, instead of just MySQL.

Fenris [5] is a fork from Bugzilla. One of the most important differences is that, instead of appending bug reports to a string blogs Fenris orders individual comments in database tables according to privilege levels in case the report reveals sensitive information. Other features include the ability to edit and delete comments, more conditional system variables than Bugzilla does as well as email hiding to protect user's privacy.

Issuezilla [11] is another fork from Bugzilla, supported by collab.net. Some Issuezilla team members are regular contributors to the Bugzilla mailing list/newsgroup. Issuezilla is not the primary focus of bug-tracking at tigris.org however. Scarab is Issuezilla's bug-tracking system built using Java Servlet technology. In addition to the standard features, Scarab has fully customizable and an unlimited number of Modules (various projects), Artifact types (Defect, Enhancement, Requirement, etc), Attributes (Operating System, Status, Priority, etc), Attribute options (P1, P2, P3) which can all be defined on a per Module basis so that each of your modules is configured for users specific tracking requirements.

The *Debian Bug Tracking System* [13] is an e-mail based system with a web-based report generator. It is in active use by the the Debian project. Initially, a bug report is submitted by a user as an ordinary mail message to submit@bugs.debian.org. This will then be given a number, acknowledged to the user, and forwarded to debian-bugs-dist. If the submitter included a Package line listing in a package with a known maintainer, the maintainer will get a copy too. Each report has a separate email address for submission of additional information. All manipulation of reports are done by email while bug-report viewing is done by the web, or via e-mail.

Request Tracker [14] is an enterprise-grade ticketing system which enables a group of people to intelligently and efficiently manage tasks, issues, and requests submitted by a community of users. Tickets can be opened by email, web or command line. Written in object oriented perl it uses a MySQL backend. RT manages key tasks such as the identification, prioritization, assignment, resolution and notification required by enterprise-critical applications including multiple project management, help desk, NOC ticketing, CRM and software development. Open Source with commercial support.

Roundup [10] is an issue tracker written in Python that can use multiple storage back-ends. It offers accessibility through the web, email, command-line or Python programs. It can be used to track bugs, features, user feed-

back, sales opportunities, milestones. Amongst interesting features Roundup provides is the possibility to write customised automatic auditors and reactors that perform actions before and after changes are made to entries in the database, or may veto the creation or modification of items in the database.

2.5 Conclusions

As can be seen from this chapter, there are a significant number of F/OSS projects with large user communities. The projects presented here could be presented in more detail, and there are certainly more projects that can be described.

The brief survey highlights that there are many aspects to a F/OSS project's code distribution process. Perhaps the most important lesson is that the process is *community-oriented*, and its success depends on how well the efforts of the community are harnessed and this in turn has a direct impact on the quality of the distribution that runs on the end-user machine.

This observation in turn has an impact on the measures, since it means that they are not purely technical. Imagine that an editor organisation like MandrakeSoft spend 50 man-months preparing a distribution release. It makes a difference if these 50 months are engineering man-months or 25-engineer and 25 community management (as could be the case in a peer-to-peer based architecture for distributing packages since the community's participation would be even more important.)

With regard to **Technical Issues**, the major F/OSS projects are quite similar. They principally use mirror servers for code distribution though there has been a definite recent trend towards peer-to-peer systems like BitTorrent. Tools for bug reporting are also currently the subject of improvements. Most projects must therefore suffer from the same problems highlighted by MandrakeSoft.

Chapter 3

Measurement and Evaluation

3.1 Introduction

This chapter specifies a methodology for the measurement and evaluation and defines a set of metrics for the *code distribution process* for Free and Open Source Software (F/OSS). Our methodology leans on the evaluation process described in the ISO/IEC 14598 standard, the process of definition of quality models described in the ISO/IEC 9126-1, as well as on some elements of the Goal/Question/Metric method.

The *methodology* consists of the following steps:

1. Identifying the purpose and goals of the measurement and evaluation
2. Specifying a quality model
3. Defining metrics
4. Establishing a measurement and evaluation plan
 - (a) Rating levels for metrics
 - (b) Criteria for evaluation
 - (c) Methods and schedule
 - (d) Simulation of data interpretation
5. Data collection - measurement, rating, evaluation
6. Interpretation

In the following, we specify the purpose and goals of measurement, a quality model and a set of metrics. The metrics will be then be used to compare different architectures for the *code distribution process*. The architectures that can be taken into consideration are:

- the current architecture used by MandrakeSoft, which consists of a hierarchy of mirror servers;
- the improved version of the architecture described above;
- one or more architectures that would be partly or completely based on peer-to-peer (P2P) systems.

The P2P systems are made up of a relatively large number of members who both contribute to and profit from the membership of the community. These systems do not require a centralized control.

3.2 Identifying the purpose and goals

We define the purpose of the measurement and evaluation in the EDOS Workpackage 4 as follows:

Purpose: COMPARE DIFFERENT ARCHITECTURES for the distribution of open source software.

In order to be effective, specific and explicitly stated goals should be specified. Each goal should be focused on a certain aspect of the code distribution process. As suggested by the ISO/IEC 14598 standard, the evaluation process can be described from different points of view, such as the point of view of editors (maintainers), developer users or end users. Developer users are those who contribute to the development of the software by testing the current software version and reporting bugs. The end users are those who are primarily interested in getting a stable version of the software.

Two **goals** are specified:

G1: Improvement of the QUALITY OF SERVICE from the point of view of DEVELOPER USERS.

G2: Minimization of the COSTS from the EDITOR's point of view.

After explicitly stating the goals, quality models with corresponding metrics need to be specified.

3.3 Specifying a quality model

A *quality model* is to be defined using ISO/IEC 9126-1 standard for software product quality as a starting point and an inspiring example, but having in mind that EDOS certainly requires different and specific quality model, since the characteristics that we wish to measure relate to the code distribution process, rather than just the software quality itself. It is fundamental to the preparation of any evaluation that a quality model reflecting the user's requirements of the objects to be evaluated is constructed. Therefore, we will focus to the goal G1 first, and create a quality model for the quality of service in the code distribution process from the point of view of the developer user.

A quality model consists of a set of quality characteristics, each of which can be decomposed into a set of quality sub-characteristics. The characteristics and sub-characteristics depend on the purpose of the evaluation. The structure is hierarchical, and, theoretically of unlimited depth. For the specified goal, i.e. improvement of the quality of service from the point of view of the developer user, the quality model is presented in figure below:

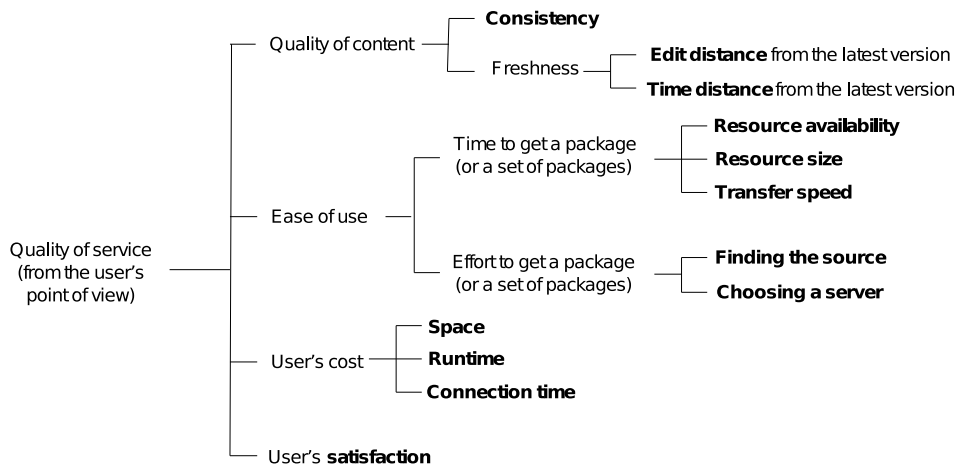


Figure 3.1: Quality model for analyzing the quality of service from the developer user's point of view

Definitions and descriptions of characteristics and sub-characteristics are given in the following. While the meaning of some characteristics is quite obvious from their name, other characteristics (like consistency and freshness) need a more detailed explanation.

A. QUALITY OF CONTENT

A.1. CONSISTENCY

Definition: The edit distance between the set of packages on the target computer and the reference set of packages as it existed at the source server in the beginning of downloading.

Description: Consistency is violated whenever a software component in one package requires software that is not present, or not up to date, in another package. As an example, imagine that an end-user installs version 1.2 of OpenOffice. The core program requires libraries that must also be installed. The installation becomes inconsistent if the libraries subsequently installed are not version 1.2. Consistency failure - the inability to locate the correct package versions - is currently the greatest complaint of Linux users. When examining consistency, the `hdlist` files can be used. In the Mandrakelinux distribution, an `hdlist` file is automatically generated on the Cooker machine every 20 minutes. The `hdlist` file contains a list of packages that are currently on the Cooker and the metadata about each package. No matter if no packages changed in this 20 minutes time interval or `n` packages changed, the `hdlist` file is consistent against the list of packages on the main server and it includes the versions of packages that are maximally 20 minutes old. The mirrors get the latest `hdlist` when they start the download. The examination of consistency includes a comparison of the list of packages on the mirror with the corresponding `hdlist` file.

Comment: The notion of "weak consistency" may also be taken into consideration. It concerns the situations when a user has a set of packages that can be compiled against each other, but this state is different from the one on the main server.

A.2. FRESHNESS

Description: It is useful to distinguish freshness and consistency since many clients may run old versions of the F/OSS distribution. It is important to measure freshness since it influences the implementation of the distribution network. For instance, if few users actually possess recent packages, then this impedes on the efficiency of a peer-to-peer based architecture. We distinguish two aspects of freshness. The first one is related to the edit distance between the distribution version on a target server (or user's client) and the version on the main server. The second one deals with the time interval needed for arrival of a package (or the complete distribution version) after publishing on the main server.

A.2.1 EDIT DISTANCE FROM THE LATEST VERSION

Definition: The edit distance between the set of packages on the target

computer and the set of packages at the main server at a certain moment.

A.2.2 TIME DISTANCE FROM THE LATEST VERSION

Definition: Time interval between the moment of publishing a new package on the main server and the moment of its arrival to a user.

In the figure below we have an example of a set of 4 packages copied from a MainServer to a MirrorServer at different moments of time:

B. EASE OF USE

B.1. TIME to get a package (or a set of packages)

B1.1. RESOURCE AVAILABILITY

Definition: The server's capability of providing users with needed packages.

B1.2. TRANSFER SPEED

Definition: Amount of data transferred to the user in a certain time interval.

B1.3. RESOURCE SIZE

Definition: The size of the package or a set of packages.

B.2. EFFORT to get a package or a set of packages

B.2.1. FINDING THE SOURCE

Definition: The effort needed to find a certain package or a set of packages.

B.2.2. CHOOSING A SERVER

Definition: The effort to choose the appropriate server for downloading.

C. USER'S COSTS

Comment: these characteristics are primarily related to P2P systems where users offer some storage space to the community and additional CPU time for P2P functioning.

C.1. SPACE

Definition: Storage space a user is willing to offer to the P2P community.

C.2. RUNTIME

Definition: CPU time a user needs for the functioning of the system during the code distribution.

C.3. CONNECTION TIME

Definition: Time a user is connected to the network.

D. USER'S SATISFACTION

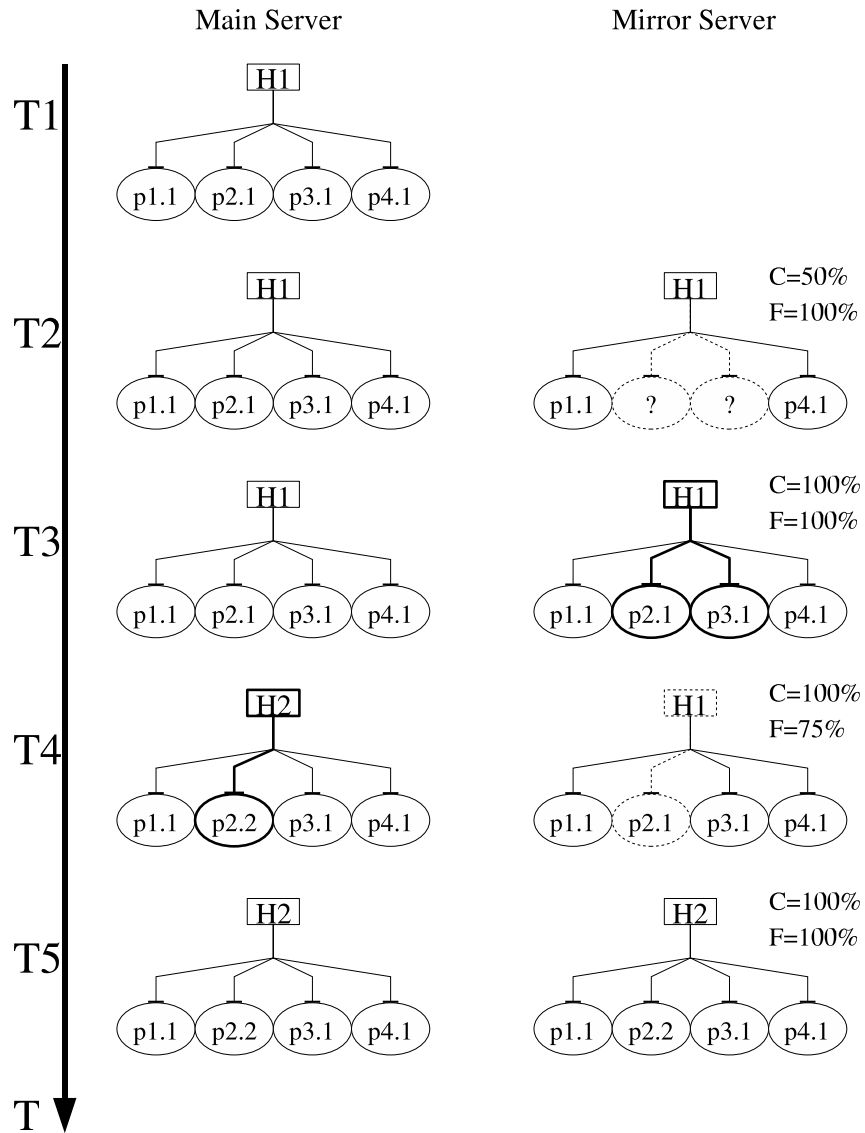


Figure 3.2: Example for Consistency and Freshness

Definition: User's satisfaction with the quality of service in the code distribution process.

Although not included into the quality model, the **security** issues should also be taken into consideration. A F/OSS distribution network is an obvious target for security attacks. The network must be able to withstand such attacks, both for the F/OSS project's prestige and for the security of content downloaded from the network. An attack is any malicious action that can interrupt the correct and continued functioning of the distribution network. An attack may lead to a *denial of service*, preventing clients from using the system. This can happen in a variety of ways, e.g., corruption of rsync scripts on mirror servers following an intrusion. Such an attack took place on Debian Linux mirror servers in November 2003. An intrusion took place on four mirror services which led to the e-mail service being shutdown and developer accounts being frozen. The attack occurred just prior to a new release being made, which ironically contained security bug fixes.

An attack can also interfere with the *integrity* of package content. The content might be corrupted between the time the package is created and its installation on an end-user machine. A corrupted package might not compile. Worse, the package can be surreptitiously modified through the addition of virus or Trojan Horse code. The mostly widely used mechanism to ensuring integrity is to have the developer or packager sign packages. RPM – the Red Hat packaging technology – allows developers to sign their packages using PGP-based digital signatures. These signatures can be used to detect attacks on integrity.

Nonetheless, there will be more to say about the security issue. For instance, the RPM approach does not handle the problem of signing keys being lost or stolen, or of rogue developers signing rogue packages that they claim to be part of a distribution. Nor is the issue of malicious – though trusted – developers adding Trojan Horses to packages.

*Comment: **Customization** can be included in the "ease of use" part of the quality model. The F/OSS users can differ greatly in their software needs, and consequently on the set of F/OSS distribution components that they install. It is useful for users to be able to customize the choice of packages to download from the distribution network in a simple fashion.*

3.4 Defining metrics

Each terminal node of the quality model hierarchy will have at least one metric associated with it. It is recommended to identify a significant and minimal set of metrics that are clearly related to the goals of the measurement activity. For some defined metrics, the measurement procedure in the current system is described. However, for many metrics it is not possible to explain the measurement procedure in detail or to give a judgement about the feasibility of the metric before different architectures are specified.

A. QUALITY OF CONTENT

A.1. CONSISTENCY

For the consistency, the following four metrics are proposed:

- M1. number of missing packages on a server for a certain distribution version
- M2. % of non-missing packages on a server for a certain distribution version
- M3. AVG number of missing packages on servers for a certain distribution version
- M4. % of non-missing packages on all servers for a certain distribution version

Procedure: For the current architecture, a hierarchy of mirrors, the procedure of getting the values for metrics M1, M2, M3 and M4 is as follows. The starting point is the mirrors list known by MandrakeSoft. As already mentioned, a `hdlist` file is created every 20 minutes on the main server and it contains a list of packages and metadata about each package. For each mirror it is checked whether the content of `hdlist` file reflects exactly the list of packages in the given directory on the mirror. This way the number of missing packages can be determined. A Perl module is used to connect to each mirror in the list and to retrieve the packages list and the `hdlist` file. All the information about mirrors' state, missing files number, etc. is stored in a database. A similar procedure can be used for any other architecture.

Comment: If we introduce weights according to the importance of different packages, then we can also have the metrics like: (AVG) number of important files missing.

Comment: There is an option to include also the notion of "weak consistency" concerning the cases when users create new consistent states different from the one on the main server. This can result with an additional set of metrics.

A.2. FRESHNESS

A.2.1 EDIT DISTANCE FROM THE LATEST VERSION

M5. Number of packages on a mirror that have the same package version number as corresponding packages (the latest version) on the main server at a certain moment of time

M6. % of packages on a mirror which have the same package version number as corresponding packages (the latest version) on the main server at a certain moment of time

Procedure: For the hierarchy of mirrors, the measurement for M5 and M6 can be done by comparing hdlist file on the mirror with the one on the main server at a certain moment of time. A similar procedure can be used for any other architecture.

A.2.2 TIME DISTANCE FROM THE LATEST VERSION

M7. Time between publishing a new package on the main server and its arrival on a particular server

M8. AVG time (for all servers) between publishing a new package on the main server and its arrival on servers

M9. Time between publishing a new package on the main server and its arrival to user

M10. AVG time between publishing a new package on the main server and its arrival to a user

M11. Time between publishing a new distribution version on the main server and arrival of the whole distribution version to user

M12. AVG time between publishing a new distribution version on the main server and arrival of the whole distribution version to user

Procedure: For the hierarchy of mirrors, the measurement for M7, M8, M9, M10, M11 and M12 includes comparisons of the timestamps on the hdlist files (one on the mirror with the one on the main server) at a certain moment of time. A similar procedure can be used for any other architecture.

Comment: The measurement procedure for some of the mentioned metrics, like M7, is already clear for the hierarchy of mirrors, and probably a similar procedure will be used for other architectures. On the other hand, it is not clear at this moment how feasible it will be to get the appropriate values for the metrics like "AVG time between publishing a new package on the main server and its arrival to a user". A simulation, which includes introducing weights and yielding equations, may be needed.

Comment: MandrakeSoft will try to draw a map of this network and the syn-

chronization protocol used between the mirrors by simulating a "traceroute" of a new .rpm package arrived on the main server.

B. EASE OF USE

B.1. TIME to get a package (or a set of packages)

B1.1. RESOURCE AVAILABILITY

M1, M2, M3, M4 (see: consistency)

M13. Time needed for one server to get synchronized with the main server

M14. AVG time needed for each server to get synchronized with the main server

Procedure: For the hierarchy of mirrors, in order to get the values of M13 and M14 for a mirror at a certain hierarchy level, we have to sum all the intermediate synchronization times between mirrors.

Comment: Process of synchronization between two servers is the process of copying the content of one server to another and having the same content as a result. The rsync protocol is used in MandrakeSoft for copying the packages.

M15. Time % availability per server

Comment: This is more important in P2P environment.

Question: Should we also include "AVG time to obtain server availability (recovery time)"?

B1.2. TRANSFER SPEED

M16. AVG (all servers) transfer speed per server

Procedure: transfer speed = resource size / download time

M17. number of concurrent users on a server

Comment: Statistics about number of clients and downloads are made by MandrakeSoft.

M18. AVG time to download a package (or a set of packages) to the user

B1.3. RESOURCE SIZE

M19. AVG package size

M20. AVG number of packages for application

B.2. EFFORT to get a package or a set of packages

B.2.1. FINDING THE SOURCE

M21. AVG time per client to find a needed package (or a set of packages)

B.2.2. CHOOSING A SERVER

Comment: No metrics for choosing a server have been yielded yet. It would be useful to have an approximation of the user's ability to choose the fastest server (mirror). In the hierarchy of mirrors, the following questions are interesting: How near is the mirror? What is the bandwidth? How many levels of mirroring there are (AVG)? Different metrics in respect to the number of levels may be defined.

*** M22. AVG time to find a package (or a set of packages) + AVG time to download a package (or a set of packages)

Comment: Metric M22 is related to both time to get a package and effort to get a package.

C. USER'S COSTS

C.1. SPACE

M23. Extra space the user is willing to offer to the community

C.2. RUNTIME

M24. % CPU time for P2P functioning

C.3. CONNECTION TIME

M25. % upload time / (upload + download time)

M26. AVG connection time of a user in respect to the number of users using a system

D. USER'S SATISFACTION

M27. number of bug reports in a fixed period

Comment: Currently, user's feedback is got by a mailing list and Bugzilla reports.

M28. AVG value of users' personal ratings?

Comment: Users can be asked some questions in order to evaluate how satisfactory is the performance from their point of view.

For the **security** issues, only a description of the risks regarding different architectures will be presented.

3.5 Future work

After proposing and specifying different architectures for the code distribution process, the feasibility of the defined metrics will be evaluated, and the set of metrics will be further refined. For each metric, it should be explained *why* the measure is important, and *how* the measurement is made for that metric.

A *measurement and evaluation plan* will then be established. The plan includes:

1. Rating levels for metrics Establishing rating levels for metrics involves determining the correspondence between the score produced on some scale and the degree of satisfaction of the requirements.
2. Criteria for evaluation Establishing criteria for evaluation involves defining a procedure for summarizing the results of the evaluation of the different characteristics, using for example weighting functions of different kinds.
3. Methods and schedule Methods and schedule for measurement and evaluation should be described. It should be defined who is responsible for collecting metrics data values.
4. Simulation of data interpretation A simulation of data interpretation should also be done before the actual measurement starts. The simulation contains expected values of metrics, graphs and charts.

Once having a plan, the measurement and evaluation process can begin. As already mentioned, measurement gives a score on the scale, rating determines the correlation between the raw score and the rating levels, and evaluation is a summary of the set of rated levels. The individual ratings are put together in order to get an overall picture, which reflects the relative importance of different characteristics in the light of the particular quality requirements. Finally, the results of the measurements are discussed and conclusions are made.

Chapter 4

Glossary

4.1 Foreword

The aim of this glossary is to clarify terms that are often used within the EDOS consortium in relation to Free and Open Source Software (F/OSS). Nonetheless, the glossary should be understandable to anyone outside of the consortium by providing a common language and understanding of the field.

The glossary is (currently) divided into two parts. Section 4.2 presents **general terms** used in EDOS. These are common terms that must be clearly understood in order to understand F/OSS. Section 4.3 defines **proper terms** from the F/OSS field, such as the names of software tools, systems, or software projects.

Notation Terms used in definitions that have their own definition in the glossary are written in the SMALL CAPS style.

4.2 General Terms

Academic Free License or **AFL**. An open source software LICENSE that contains a clause revoking the license if a third party asserts a patent against any package of the software. Like the BSD LICENSE, the AFL does not prohibit the use of licensed material in products for resale.

AFL See ACADEMIC FREE LICENSE.

Alpha Release The first RELEASE of software made by a F/OSS PROJECT.

Archive A single large file that packs together multiple files. The contents of an archive are usually compressed in order to save space. Common archival formats are TAR, GZIP and ZIP.

Benevolent Dictatorship Development-control structure of FREE AND OPEN SOURCE SOFTWARE. When a PROJECT starts attracting enough voluntary contributors, an individual PROGRAMMER or a team of programmers takes on the role of gatekeeper, and makes the final decision about what contributions are added to the CODE BASE. The particularity of this structure is that anyone in the community can take the code base and start adding his own modifications if the benevolent dictator is no longer seen to act in the interests of the community. This results in a PROJECT FORK under the control of a different benevolent dictator.

Beta Release The second RELEASE of software made by a F/OSS PROJECT.

Binary Code A program that has been compiled into machine executable form; also known as OBJECT CODE.

Binary Package A PACKAGE whose contents, once installed on an end-user machine, can be directly used without any pre-processing. Opposite of SOURCE PACKAGE.

BSD License An OSS software LICENSE that does not prohibit the use of licensed material in products for resale. It allows redistribution and modification of SOURCE CODE and redistribution of BINARY CODE as long as the license contains a copyright notice and a no-endorsement clause.

CLI See COMMAND LINE INTERFACE.

Closed Source Software Opposite of OPEN SOURCE SOFTWARE. This refers to any program distributed in BINARY CODE only form, i.e. there is no distribution of the program's SOURCE CODE as this could be a company's trade secret. As such, people who desire access to the source code are requested to sign a non-disclosure agreement in order to gain access.

Code Base Reference SOURCE CODE for a PROJECT. Contains different project RELEASES. Modifications to the source code are examined by the committers of the project, whose role is to decide if the modifications are worthy of being added to the code base.

Code Repository Central place where one or more CODE BASES are kept and maintained in an organized way.

Command Line Interface Method of interacting with a computer by sending lines of textual commands, either from a keyboard or from a SCRIPT. Contrasted with GRAPHICAL USER INTERFACE.

Committer A project participant who has the right to modify the CODE BASE of a project.

Compilation The action of translating program source code into a usable software, executable on a computer.

Compiler A computer program used to compile source code. See COMPILATION.

Copyleft The legal obligation on any person who obtains or uses F/OSS not to distribute the software for a fee – if this contradicts the LICENSE of the software – or to remove a reference to the original AUTHOR from the program.

Copyright The legal right granted to the author of a program to exclusive publication, production, sale, or distribution of the given program. (In contrast, a PATENT protects the ideas, e.g., algorithms behind the program.)

Dependency A relationship between a package and a set of packages that prevent the correct INSTALLATION, COMPILATION or running of the former package without the presence of the latter packages.

Desktop Environment A Graphical User Interface (GUI) environment to operate an end-user machine. It provides multiple features like icons,

toolbars, applications, etc. Examples of Desktop Environments include GNOME and KDE.

Developer A project participant who proposes changes to a project's SOURCE CODE. He does not have the right to make changes to the project's CODE REPOSITORY; see COMMITTER.

Disk Image An ISO Image of a CD-ROM that respects the ISO 9660 standard.

Distribution A packaging of a software project's SOURCE CODE, BINARY CODE and documentation with various user interfaces, utilities, and other software into a user deliverable. The term is often used in relation to the Unix and Linux operating systems.

Editor A group or organisation that undertakes the task of packaging software components into a DISTRIBUTION and issuing a RELEASE.

Feed A Web site that acts as a source of PACKAGES, PATCHES or documentation for a PROJECT.

F/OSS See FREE AND OPEN SOURCE SOFTWARE.

F/OSS Process A series of operations performed by a number of individuals in the production of F/OSS. The operations can be automatic, like the delivery of packages to end-user machines, or manual, like the election of developers to the status of committers.

Free Software License Type of LICENSE describing software distributed in a BINARY CODE form only, that can be used at no cost. PROPRIETARY software can be free software. The SOFTWARE AUTHOR can specify usage restrictions in the license.

Free and Open Source Software or **F/OSS**. Software that is free or distributed with an OPEN SOURCE license. This is the class of software that the EDOS project deals with.

Freeware See FREE SOFTWARE LICENSE.

GNU General Public License See GPL LICENSE.

GNU Lesser General Public License See LGPL LICENSE.

GPL License or **Gnu General Public License**. A software LICENSE often associated with OPEN SOURCE SOFTWARE. This license stipulates that any user may read and make changes to the software, and that resulting software must be licensed under the GPL too. The goal is to force people willing to develop CLOSED SOURCE SOFTWARE to reinvent the wheel, and prevent them from taking advantage of the community's effort.

Graphical User Interface Method of interacting with a computer through manipulation of graphical components. Examples of components are buttons and windows that can be clicked with a cursor. Contrasted with COMMAND LINE INTERFACE.

GUI See GRAPHICAL USER INTERFACE.

Intellectual Property Rights or **IPR**. The broad issue of defining who may specify conditions on the use and distribution of a SOFTWARE.

IPR See INTELLECTUAL PROPERTY RIGHTS.

Installation The process of transforming a set of packages into source code, binary code and documentation files on an end-user machine that can be, respectively, modified, run and browsed by the end-user.

Issue Tracking The process of communicating user satisfaction, bug reports, new program requirements from end-users to the PROJECT DEVELOPERS.

ISO 9660 A standard that defines a file system for CD-ROM media and that is currently supported by the major OS producers. See DISK IMAGE.

ISO Image A file containing the complete content and structure of a CD-ROM. Often used to let people burn remotely a copy of an original CD-ROM.

Kernel The core of an operating system, the part whose presence is mandatory for core programs to run. Common examples include HURD and the LINUX KERNEL.

License A formal agreement between a software producer and end-user that stipulates the conditions the end-user must respect to gain access to the software, as well as the restrictions imposed on the copy of the software to third parties.

LGPL License or **GNU Lesser General Public License** (formerly GNU Library General Public License). A LICENSE that places a COPYLEFT restriction on SOURCE CODE files but not on the computer program as a whole, provided the computer program in which the LGPLed source code is included follows some guidelines. The LGPL license is a compromise between the GNU GENERAL PUBLIC LICENSE and simple permissive licenses such as the BSD LICENSE. As a result, the key difference between the GPL LICENSE and the LGPL LICENSE is that LGPLed source code can be linked to non GPL or non LGPLed software such as free or proprietary software.

Meritocracy A social system in which power is proportional to merit. As a developer acquires merit through the value of his contributions, he obtains new rights like committing code, and more weight is attributed to his opinion in project votes. This ‘the more you do the more you are allowed to do’ system is used by some open source projects, like Apache.

Object Code See BINARY CODE.

Open Source Software or **OSS**. A collective name for all kinds of software under a LICENSE defining that their SOURCE CODE is accessible. Anyone may extend, improve and distribute Open Source Software as long as this respects the clauses of the licence. Opposite of CLOSED SOURCE SOFTWARE.

OSS See OPEN SOURCE SOFTWARE.

Package A single file that contains source code files, machine binary files or documentation files. It may also contain scripts that, when run, install the files on the user’s machine. Package meta-data include version numbers, date of creation and a signed checksum to verify package integrity.

Patch An update or bug fix for SOFTWARE. It is either a program that modifies the original BINARY CODE, or a list of instructions to be followed by a tool to modify the original SOURCE CODE.

Patent A certification made by a government authority that confers upon the creator of an invention, or idea, the exclusive right to make, use, and sell that invention for a set period of time.

Port A collection of files along with automated tools for installing, compiling and updating these files. The ports system was originally developed for FreeBSD, and is an alternative to PACKAGES.

Progressive Open Source The application of F/OSS principles within the confines of a closed community, like that of a university or enterprise.

Project An undertaking by one or more individuals to develop and maintain F/OSS with specific software requirements.

Project Fork The birth of a new PROJECT from an existing project. The new project may be motivated by a schism between the individuals of the parent project, or simply to accommodate a new set of software requirements.

Public Domain Software Software that can be distributed free of charge, without contradicting any LICENSE, COPYRIGHT or PATENT.

RC See RELEASE CANDIDATE.

Release A frozen state of the project's software and documentation that is made available to end-users.

Release Candidate or **RC**. A state of the project's software and documentation that is considered by the project's management for release.

Release Management An aspect of the F/OSS process that covers the delivery of software to end users. Issues include versioning, level of accepted stability and deadline setting.

Revision Control or VERSION CONTROL. Content control system whose mission is to keep track of all information concerning the evolution of a PROJECT and especially its SOURCE CODE. The files are kept in a repository and each modification applied to them increments the REVISION LEVEL. Key features provided by revision control systems are the ability to recover an old revision of the contents, create different branches for different evolutions of the project or the ability to freeze branches (for instance to prepare a RELEASE).

Revision Level VERSION number associated with a file stored in a REVISION CONTROL system.

Script A sequence of command line instructions that can be invoked and run as a unit, rather than having to invoke one instruction after another.

Shareware Proprietary software that is distributed on a free basis for a trial period or with restricted functionalities. Paying for it allows the use of the software beyond the trial period and give access to all functionalities.

Software A computer program, or group of programs, in SOURCE CODE or BINARY CODE form.

Software Author A person or group of people who create a program and then distribute it under a chosen LICENSE.

Source Code Computer program code in "human readable" form written by DEVELOPERS. Opposite to BINARY or OBJECT code. Source code is subject to CODE VERSIONING.

Source Package A PACKAGE whose contents, once installed on an end-user machine, require processing prior to use. Examples of processing include source code compilation, or the generation of man pages from contained documentation files.

Version Software unique identifying number. Versions of SOURCE CODE can be maintained using a REVISION CONTROL system. Once the development team considers a software version as being sufficiently mature, the software version can be turned into a software RELEASE.

Version Control see REVISION CONTROL.

4.3 Proper Terms

Apache A short form for *Apache Software Foundation*. This foundation has a number of open-source projects in the area of Web services, and has its own software LICENSE. The foundation was formed in 1999 from the group who developed the Apache HTTP server.

BitKeeper A distributed software development tool that runs on all major OS platforms. It includes CODE BASE management tools such as REVISION CONTROL.

BitTorrent A peer-to-peer based content distribution system that is adapted to very large files. Its distinguishing characteristic is that it downloads different chunks of a file to different users; these users can then exchange chunks in a peer-to-peer fashion. This approach allows the content distribution network load to be distributed across the network.

BSD for *Berkeley Software Distribution*. This is a UNIX distribution from the University of California at Berkeley. The first version appeared in 1975.

Bugzilla This is a Mozilla project whose aim is the development and maintenance of a ISSUE TRACKING system. Its features include a MySQL back-end for the bug repository, a white board to which one can post attachments and a bug definition language.

Concurrent Versions System or **CVS**. A file repository REVISION CONTROL system with optimistic concurrency control. Older versions of files are maintained so that rollbacks may be effected.

Cooker See MANDRAKE COOKER

CVS See CONCURRENT VERSIONS SYSTEM.

Debian A community project whose goal is to furnish a complete operating system based on the Linux kernel initially, though efforts are underway to use the HURD kernel. The project was formed in 1993.

Dpkg or *Debian Package Manager*. A packaging and installation tool for Internet downloads, included with Debian Linux but compatible with other Linux distributions. It produces files with a .DEB extension. Similar to RPM.

Eclipse An open-source project managed by the Eclipse Foundation that deals with tools for development. Several programming languages are supported and developments mainly centre around the popular Eclipse development environment.

Fedora A Linux project that stemmed from RED HAT in 2003. Fedora is a complete operating system built exclusively from free software.

Free Software Foundation or **FSF**. An organisation founded in 1983 by Richard Stallman. The foundation advocates that software SOURCE CODE should be accessible to anyone, who may then propose and make improvements to the code, and even redistribute it.

FreeBSD A project based on the Berkeley Software Distribution version of Unix. Related projects are OpenBSD and NetBSD.

FSF See FREE SOFTWARE FOUNDATION.

Gentoo A free Linux-based operating system that allows end-users a large degree of control over the installation process. This is achieved using a PORTS SYSTEM similar to that used by the BSD operating systems OpenBSD and FreeBSD.

GNATS GNATS is a GNU project toolset for tracking bugs reported by users to a central site. The client front-end can be based on a COMMAND LINE, Emacs or a Web interface.

GNOME This is the GNU Network Object Model Environment. It is a GUI for desktop applications originally intended for the Linux operating system, but which also runs on any Unix platform.

GNU An acronym for *Gnu is Not Unix*. This is a free software project created in 1984 whose goal is to develop and maintain a free version of the Unix operating system.

Hurd A GNU project whose aim is to develop a replacement for the Unix kernel. The system is structured as a set of servers running over the Mach micro-kernel.

Issuezilla This is a fork from the Bugzilla project that is managed by colab.net. It is based on Java Servlet technology and has a more elaborate bug description language than BUGZILLA.

KDE The **K** Desktop Environment is an OPEN SOURCE graphical DESKTOP ENVIRONMENT for Unix workstations. The project was launched by Matthias Ettrich in 1996; his aim was to make Unix computing more easy to use.

Knoppix A bootable CD-ROM containing a complete Debian distribution. This ISO image can also be downloaded from the Knoppix Web site. The aim of the project is to automate the installation of the system on an end-user machine.

Linux A Unix-like operating system developed especially for the Personal Computer market. It is composed of the LINUX KERNEL and systems programs from the GNU project.

Linux Kernel An operating system kernel based on Unix that was developed by Linus Torvalds. The kernel is an OPEN SOURCE SOFTWARE project with Torvalds as BENEVOLENT DICTATOR. Several DISTRIBUTIONS based on this kernel exist.

Mandrake Cooker The development version of the next Mandrakelinux release. It allows both users and DEVELOPERS to improve the current code by signaling bugs, correcting them, or adding new features. This version is up-dated all the time and is therefore not stable.

Mandrakelinux A Linux DISTRIBUTION produced by Mandrakesoft that specialises in easy-to-use front-ends for both home and office users.

Mozilla An OPEN SOURCE SOFTWARE project that specialises in Internet tools, e.g., Web and mail clients, bug reporting (BUGZILLA), ISSUE TRACKING and project testing (Tinderbox).

OOo See OPENOFFICE.ORG.

OpenBSD A variant of the Unix BSD operating system. This variant supports binary compatibility with programs from Solaris, FreeBSD, and Linux.

OpenOffice.org An open source project whose goal is to develop and maintain a suite of Office applications that are compatible with the Microsoft Office suite. OpenOffice is an off-shoot of StarOffice; the latter is a product that was bought by Sun Microsystems in 1999.

Red Hat A former Linux DISTRIBUTION that split into forked to FEDORA and RED HAT ENTERPRISE in 2001. Red Hat is actually a company

and its distribution was one of the first for Linux. The distribution is also famous for its packaging technology RPM.

Red Hat Enterprise A proprietary system for the managed distribution of Red Hat Linux. Customers pay for the distribution network which is tailored to the needs of its clients. A client can be an end-user, a developer, beta-tester or any combination of these.

Request Tracker A ticketing system for a community project that supports project management, ISSUE TRACKING, help desk, CRM and software development. The system is platform independent and open-source.

Rsync A tool used for incremental file transfer between machines. It is an open-source tool that is distributed with a GNU license. The tool is used by many F/OSS projects to copy RELEASES between mirror servers.

Slackware A Linux distribution that seeks to be as close to Unix as possible.

Bibliography

- [1] FreeBSD handbook, http://www.freebsd.org/doc/en_us.iso8859-1/books/handbook/index.html, Jan. 2005.
- [2] FreeBSD porter's handbook, http://www.freebsd.org/doc/en_us.iso8859-1/books/porters-handbook/, Jan. 2005.
- [3] Bugzilla. <http://www.mozilla.org/bugs/>, Jan. 2005.
- [4] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.
- [5] Fenris. <http://www.lokigames.com/development/fenris.php3>, Jan. 2005.
- [6] S. Ghemawat, H. Gobioff, and L. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 29–43, Bolton Landing, NY, USA, Oct. 2003. ACM.
- [7] GNATS. Gnats: The gnu bug tracking system, <http://www.gnu.org/software/gnats>, Jan. 2005.
- [8] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, volume 37, 5 of *Operating Systems Review*, pages 314–329, New York, Oct. 19–22 2003. ACM Press.
- [9] B. RedHat. <http://bugzilla.redhat.com/bugzilla/>, Jan. 2005.
- [10] Roundup. <http://roundup.sourceforge.net/>, Jan. 2005.
- [11] Scarab. <http://scarab.tigris.org>, Jan. 2005.

- [12] M. Stokely. FreeBSD release engineering, http://www.freebsd.org/doc/en_us.iso8859-1/articles/releeng/index.html, Jan. 2005.
- [13] D. B. T. System. <http://www.debian.org/bugs/>, Jan. 2005.
- [14] B. P. R. Tracker. <http://www.bestpractical.com/rt/>, Jan. 2005.
- [15] S. Witty. Best Practices for Deploying and Managing Linux with Red Hat Network.