Measuring Etch: the size of Debian 4.0*

Juan José Amor, Gregorio Robles, Jesus M. González-Barahona, Javier Fernández-Sanguino Peña
GSyC/Libresoft, Universidad Rey Juan Carlos (Madrid, Spain)
{jjamor,grex,jgb,jfs}@gsyc.escet.urjc.es

Abstract

The Debian GNU/Linux operating system is one of the most popular GNU/Linux distributions, not only among end users but also as a basis for other distributions and embedded systems. Besides being popular, it is also one of the largest software compilations and thus a good starting point to analyze the current state of libre (free, open source) software. This work is a preliminary study about the new Debian GNU/Linux release (4.0, codenamed etch), which was officially released April 8th, 2007. We have measured the size of Debian in lines of code (which is close to 283 million source lines of code), what programming languages have been used to write that code, and the size of the packages included in the distribution. We have also applied a 'classical' and well-known cost estimation methods which gives an idea of how much it would suppose to create something as big as Debian from scratch. We estimate that cost to be over 10 billion USD.

1 Introduction

On April 8th 2007, the Debian Project announced the official release of the Debian GNU/Linux version 4.1, codenamed *etch*, after 21 months of development [8]. The Debian distribution is put together by the Debian project, a group of more than 1,000 volunteers (also known as maintainers) whose main tasks cover picking up libre software packages, adapting, integrating with other pieces of the operating system and packaging them for inclusion in the distribution [18]. Debian maintainers package software which they obtain from the original (upstream) authors, ensuring that it works smoothly with the rest of the programs in the Debian system. All packages have to follow the technical requirements defined in the *Debian Policy Manual* [7] which ensures that they interoperate properly with other packages and follow the standards the operating system follows, which include the *Filesystem Hierarchy Standard* (or FHS) and, partially, the *Linux Standard Base* (or LSB, submitted as ISO/IEC 23360).

Debian 4.0 includes all the major libre software packages available at the time of its release. Only in its main distribution, composed just of libre software (according to the Debian Free Software Guidelines [5], or DFSG), there are more than 10,100 source packages. The whole release comprises almost 18,000 binary packages, which users can install easily from various media or via the Internet.

In this paper, we have analyzed this operating system, measured its size, and compared it to other contemporary GNU/Linux distributions. We decided to write this paper as an update of others published by our group, in the same line, starting with *Counting Potatoes* [11], and updated until *Measuring Sarge* [1], which were written to cover previous Debian releases. The structure of this paper is as follows: the next section briefly presents the methods we have used for collecting the data shown in this paper. Later on, we offer the results of counting Debian 4.0 (including total counts, counts by programming language, counts for the largest packages, etc.). The following section offers some comments on the numbers obtained and how they should be understood, as well as some comparisons with Red Hat Linux distributions and other operating systems, both free and proprietary. The paper finishes with some conclusions and references.

2 Some background about Debian

The Debian 4.0 GNU/Linux distribution is put together and maintained by the Debian project. In this section, we offer some background data about Debian as a project, and about the Debian 4.0 release.

2.1 The Debian Project

The Debian Project is a worldwide group of volunteers who endeavor to produce an operating system distribution that is composed entirely of free (libre) software. The principle product of the project to date is the Debian GNU/Linux software distribution, which includes the Linux operating system kernel, and thousands of prepackaged applications.

This distribution is available for several architectures, including Intel x86, ARM, PowerPC, MIPS, Alpha, and SPARC. Although the kernel of the Debian GNU/Linux software distribution is the Linux kernel, there are other versions, currently in pre-testing phase, that are based on different kernels like Hurd and FreeBSD's. There is also a derived distribution (Nexenta) which uses Sun's OpenSolaris kernel.

The core of the Debian distribution (called section *main*, which amounts for the vast majority of the packages) is composed only of free software, according to the DFSG. It is available on the Net for download, and many redistributors sell it on CDs or other media. The Debian distribution is put together by the Debian project, a group of over 1,400 volunteer developers spread around the world, collaborating via the Internet. The work done by those developers includes adapting and packaging all the software included in the distribution, maintaining several Internet-based services (web site, on-line archive and mirrors, bug tracking system, support and development mail lists, wiki, etc.), translate and internationalize Debian-specific content, developing tools specific to Debian (including its package management tools and installation software), and in a wide sense, maintaing all the infrastructure that makes the Debian distribution possible.

Debian developers package software which they obtain from the original (upstream) authors, ensuring that it works smoothly with the rest of the programs in the Debian system. All packages must follow the Debian Policy Manual [7]. Most of the work associated with the packaging a given program usually revolves around making it compliant with those rules. Developers also take bug reports, try to fix them (reporting fixes and problems upstream), follow new upstream developments, and build all the software glue needed for making Debian system work. Bugs and security holes are discussed openly, and updates fixing important problems are made available for stable releases on a daily basis so that users can maintain their systems secure and as bug-free as possible. In some cases Debian developers are themsleves the original authors of software packages for the Debian distribution, either because it was written specifically for Debian, because the upstream author is no longer maintaining the package and it is still being maintained, fixed and improved in Debian, or because the Debian version of a program is so heavily modified in Debian that it is maintained separately from the original author (and is, in essence, a fork).

Debian is unique for many reasons. Its dedication to free software, its non-profit nature, and its open development model (where most of the discussions are addressed openly in public lists) are remarkable. The project is committed to free software, as is reflected in the Debian Social Contract [6]. The definition of what Debian understand as free software can be found in the Debian Free Software Guidelines (DFSG), and in essence is the same software which can be considered *open source* software (which is not strange, since the Open Source Definition was actually derived from the DFSG).

Debian also inspires some other interesting distributions. Two of the most known derivatives are Knoppix [14], a *live* distribution, and Ubuntu [4], a distribution with a tighter release schedule (every six months) very oriented towards the end-user and sponsored up by a company which has contracted many well known Debian developers. There are other derivatives, like Metadistros [13], which have a broader scope.

2.2 Debian Etch

Debian 4.0 (etch) is the *stable* Debian release since April 2007. It includes all the major libre software packages available at that time.

In addition to *etch*, the Debian archives include two under-development releases: *testing* and *unstable*. After a freeze of the five months in which only packages fixing bugs were included in the *testing* release, development started in this release after *etch* was published. The codename of the new release will be *lenny*. The unstable release (which is always known as *sid*) is used by developers to upload new package versions (which includes new versions of the package software), and test them. In order to guarantee that the *testing* release is always in a releasable state, changes to packages need to be uploaded to *unstable*. If no important bugs are filed against the new packages nor to the packages they depend on for

an specific time frame, the packages are automatically included in *testing* and considered for inclusing in the next stable release.

Historically, Debian was split in two archives: the *regular* one and the *non-US* archive. The non-US archive included those packages which had some legal impediment to be exported from the United States of America (usually related with strong cryptography). However, since Debian 3.1, these archives are no longer used and now all packages are available in the *regular* archive.

The archive is composed of several *categories*: `main`, `contrib` and `non-free`. The `main` category is made up of packages that comply with the DFSG and do not require any non-DFSG packages. On the other side, `non-free` is the category made up of packages not compliant with the DFSG (but can be distributed by Debian) or are encumbered by patents or other legal issues that make their distribution problematic. Finally, `contrib` is the category made up of packages that comply with the DFSG but require a non-DFSG packages (that is, in `non-free`).

For the work referenced in this paper we have considered only the `main` category. It is (by far) the largest fraction of the archive, and is only composed of free software. In many respects, it is one of the largest coordinated collection of free software available on the Internet.

3 Methodology

The approach used for collecting the data presented in this paper is, in summary, as follows:

1. Where is the source code of a Debian release?

   Source code and binary packages for current distributions (those that are supported by the project) are available from the Internet in primary mirrors located at 35 different countries as well as a large number of secondary mirrors. There are over 300 mirror sites distributing all this content. Fortunately enough, source code for both current and past Debian releases is archived, and available for everyone on the Internet at `ftp://archive.debian.org` and its mirrors. The only problem is to determine the specific list of source packages for any given release, and where to access them.

2. Downloading and collecting the data

   Once we know what files to download, we have to download all of them before being able of gathering data. Since the size of the unpackaged sources for a Debian release is very large, we chose to work on a per-package basis, gathering all the relevant data from it (mainly, the number of lines of code) before removing it and following on with the download and analysis of the next one.

3. Final analysis

   Analyze the collected data and get some statistics regarding the total number of physical SLOC of the release, the SLOC for each package, the SLOC for each of several programming languages considered, etc.

In the following sections these three steps are described in more detail.

3.1 What source code makes a Debian release?

The Debian packaging system considers two kind of packages: source packages and binary packages. Binary packages provide the programs compiled for the set of computer architectures Debian supports, although there is also code which is architecture-independant that is released as a binary package (one package servers all architectures in this case). One or more binary packages can be automatically built from one source package. For example, a source package of a program can be divided in a binary package for the program itself, a binary package for its libraries (if they can be reused by other programs) and a binary package for its documentation. For this paper, only source packages are relevant, and therefore we will no longer refer to binary packages.

When building a source package, a Debian developer starts with the *original* source (ftp site, etc) of a piece of software. In Debian parlance, that source is called *upstream*. The Debian developer patches upstream sources if needed, and creates a directory `debian` with all the Debian configuration files (including data needed to build the binary package). Then, the source package is built, usually (but not always) consisting of three files: the upstream sources (a `tar.gz` file), the patches to get the Debian source directory (a `diff.gz` file, including both patches to upstream sources and the `debian` directory), and a description file (with `dsc` extension). However `dsc` files are present only in the last releases. On the other hand patch files are not present for *native* source packages (those developed for Debian, with no upstream sources).

Source packages of current Debian releases are part of the Debian archive. For every release, they reside in the `source` directory. There are sites in

the Internet including the source packages for every official Debian release to date (usually, mirrors of `ftp://archive.debian.org`). Since Debian 2.0, for every release a `Sources.gz` file is present in the `source` directory, with information about the source packages for the release, including the files that compose each package. This is the information we use to determine which source packages, and which files, have to be considered for Debian 4.0.

However, not all packages in `Sources.gz` should be analyzed when counting lines of code. The main reason to do this is the existance, in some cases, of several versions of the same piece of software. For instance, in Debian 4.0 we can find source packages for `gcc2.95`, `gcc3.3`, `gcc3.4` and `gcc-4.0`. Counting all of these packages would imply counting the GNU Compiler Collection four times, which is not the intended procedure. Therefore, a manual inspection of the list of packages is needed for every release, detecting those which are essentially versions of the same software, and choosing one *representative* for each family of versions.

These cases may cause an underestimation of the number of lines of the release, since different versions of the same package may share a lot of code, but not all (consider for instance PHP4 and PHP3, with the former being an almost complete rewrite of the latter). However, we think this effect is negligible, and compensated with some overestimations (see below).

In other cases, we have decided to analyze packages which may have significant quantities of code in common. This is the case, for instance, of `emacs` and `xemacs`. Being the latter a code fork of the former, both share a good quantity of lines which, even when not being exactly equal, are evolutions of the same *ancestors*. Other similar case is `gcc` and `gnat`. The latter, an Ada compiler, is built upon the former (originally the C compiler), adding many patches and lots of new code. In those cases, we have considered that the code is different enough to consider them as separate packages. This probably leads to some overestimation of the number of lines of code of the release.

The final result of this step is the list of packages (and the files composing them) that we consider for analyzing the size of a Debian release. This list is done by hand (with the help of some simple scripts) for each release.

3.2 Downloading and collecting the data

Once the packages and files composing Debian 4.0 are determined, they are downloaded from the Internet. Some Python scripts where used to automate this process, which (for each package) consists of the following tasks:

- Download the files that compose the package
- Extract the source directory corresponding to the upstream package (by exploding the `tar.gz` file). After extraction, data about this upstream source is gathered.
- Patch the upstream directory with the `diff.gz` file, to get the Debian source directory. After extraction, data about it is gathered.
- Delete the `debian` directory, to avoid counting maintainer scripts (stored in this directory), and gathering data about this sans-debian Debian source package.

Not all packages have upstream version. Therefore, during this process, some care has to be taken to differentiate this situations. Also, some upstream packages contain new packaged files. We need to unpack them also.

The fetching of data is done using `SLOCCount` scripts [20], three times for each package (one in each phase, see above), which stores the count of lines of code for each package in a separate directory, ready for later inspection and reporting.

The reason for fetching data three times for every package is to analyze the impact of the Debian developer on the source package. This impact can be in the form of patches to the source (usually to make it more stable and secure, to conform to Debian installation policy, or to add some functionality to it) or in the sense of installation scripts (which can be singled out when counting sans-debian source packages).

The final result of this step is the collection of all the data fetched from the downloaded packages, organized by package, and ready to be analyzed. These data consist mainly of lists of files and line counts for them, split by language.

3.3 Final analysis

The last step is the generation of reports, using `SLOCCount` and some scripts, to study the gathered data. Since in this step all the fetched data is available locally, and in a form easy to parse, the analysis can be done quickly, and can be repeated easily, looking for different kinds of information.

The final result of this step is a set or reports and statistical analysis, using the data fetched in the previous step, and considering them from different points of view. These results are presented in the following section.

4 Results of counting Debian Etch

The main results of our current analysis of the Debian 4.0 GNU/Linux release can be organized in the following categories:

- Size of Debian Etch.
- Importance of the most used programming languages.
- Analysis of the evolution in the size of the most relevant packages.
- Effort estimations.

4.1 Size of Debian Etch

We have counted the number of source lines of code of Debian GNU/Linux 4.0 in three different ways, with the following results (all numbers are approximate, see [17] for details):

- Count of upstream packages *as such*: *260,000,000* SLOC.
- Count of Debian source packages: *283,000,000*.
- Count of Debian source packages without `debian` directory: *277,000,000*.

For details on the meaning of each category, the reader may revisit the section 3.2. In short, the count of upstream packages could be considered as the size of the original software used in Debian. The count of Debian source packages represents the amount of code actually present in the Debian 4.0 release, including both the work of the original authors and the work of Debian developers. This latter work includes Debian-related scripts and patches. Patches can be the work of Debian developers (for instance to adapt a package to the Debian policy or to locally fix a but), be provided by upstream authors (in versions which are not released yet) or might have been provided by other users (submitted as bug reports, available on the Internet) or developers of other distributions which also ship the same software. The count of Debian packages without the `debian` directory excludes Debian-related scripts, and therefore is a good measure of the size of the packages as they are found in Debian, excluding the specific Debian-related scripts.

It is also important to notice that packages developed specifically for Debian have usually no upstream source package. This is, for instance the case of `apt` and `dpkg`, package management tools specific to Debian, which are present only in the archive as a Debian source package.

4.2 Programming languages

The number of physical SLOC, classified by programming language, are (roughly rounded) as follows (numbers for Debian source packages):

- C: *145,278,000* SLOC (51%)
- C++: *52,983,000* SLOC (18.7%)
- Shell: *29,327,000* SLOC (10.4%)
- Java: *8,969,000* SLOC (3.17%)
- PERL: *8,074,000* SLOC (2.85%)
- LISP: *7,659,000* SLOC (2.7%)
- Python: *7,219,000* SLOC (2.55%)
- Assembler: *4,121,000* SLOC (1.46%)
- PHP: *3,270,000* SLOC (1.15%)
- FORTRAN: *2,678,000* SLOC (0.95%)
- C#: *2,336,000* SLOC (0.83%)
- Pascal: *2,240,000* SLOC (0.79%)
- TCL: *1,635,000* SLOC (0.58%)

Below 0.5% we find some other languages: Ada (0.46%), ML (0.42%), Objective C (0.39%), YACC (0.25%), and other below 0.1%.

In figure 1 we can view the importance of main languages versus the rest. It is consistent with the fact that most packages are written in C. C++ is another language present in most packages, and it is the main language in some of them (such as Openoffice.org or Iceweasel). The same fact occurs with Lisp, which is the main language in several packages (such as Emacs), but is also used in most of rest packages. Next, shell scripts are used to support configuration and other auxiliary task in most packages.

When we count the lines in the Debian source packages without the `debian` directory (which contains

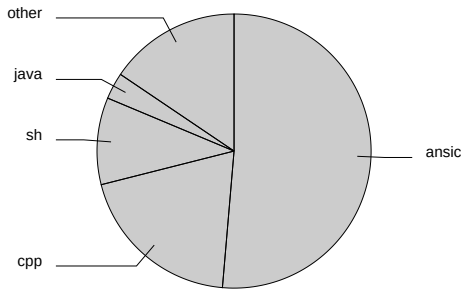

Figure 1. Pie with the distribution of source lines of code for the majoritary languages in Debian 4.0

package configuration files and maintainer scripts), the numbers are similar. This means that the maintainer scripts are not a significant part of the distribution. The main difference is in Shell and Perl lines, which uncovers the preferred languages for those scripts.

However, when we count original (upstream) source packages there are more remarkable differences. These differences can usually be amounted to patches to upstream packages made by the Debian developer. Therefore, looking at this numbers, we are able to know in which languages most patched packages are written.

4.3 The largest packages

The largest packages in the Debian etch distribution are:

- **Openoffice.org**: 5,215,000 SLOC. C++ amounts 4,613,000, while Java amounts 381,000 and C 117,000. In this package, there are representation of other 14 languages. Openoffice.org is the well known office suite, which development is leaded by Sun Microsystems.

- **Linux2.6**: 4,921,000 SLOC, mainly C code (4,700,000) and small amounts on other 10 languages. This is currently the only operating system kernel used in Debian.

- **ia32-libs**: 4,006,000 SLOC, mainly C code (3,530,000) and other 20 languages. This package contains runtime libraries for the ia32 architecture, for use on systems running 64-bit kernels.

- **gcc-4.1**: 3,630,000 SLOC, composed mainly of C lines (1,211,000) and 18 more languages. It is interesting to see that there have been identified more than 1,000,000 SLOC of shell code. This package is one of the releases of the GNU Compiler Collection included in Debian.

- **iceweasel**: 2,777,000 SLOC, mainly C++ (1,784,000). This package is a rebranded version of the Mozilla Firefox web browser. Its name is changed because of trademark restrictions imposed upstream.

- **icedove**: This package, a rebranded version of the Mozilla Thunderbird e-mail client , amounts 2,709,000 SLOC, mainly composed of C++ code (1,722,000 SLOC), C code (889,000 SLOC) and 15 more languages.

- **vnc4**: 2,357,000 SLOC, mainly C code (2,205,000 SLOC). VNC4 is a well known remote graphical console access system.

- **eclipse**: Eclipse is the extensible tool platform and Java IDE, created by IBM. It amounts 2,214,000 SLOC, mainly made of Java code (2,107,000 SLOC).

- **stalin**: It amounts 1,885,000 SLOC, mainly C (1,786,204). Stalin is a Scheme compiler.

- **mono**: Mono is a platform for running and developing applications based on the ECMA/ISO Standards, leaded by Novell. It amounts 1,766,000 SLOC and composed mainly of C# code (1,496,000 SLOC) and a remarkable amount of C lines (249,000 SLOC).

The numbers shown above are the approximage number of SLOC of the Debian source packages. Only data for the more relevant languages found in each package are reported.

The classification could be different had Debian developers packaged things in a different way. For instance, if all Emacs extensions had been included in a single Emacs package, it would have been much larger (and it have been in this *top ten* list). However, a Debian source package usually matches well with what upstream authors consider as a package, and with the general idea about what is a package.

Figure 2 shows the distribution of size in all packages for Debian 4.0. The means size can be graphically observed, and measured as 28,000 bytes in Debian 4.0. It is

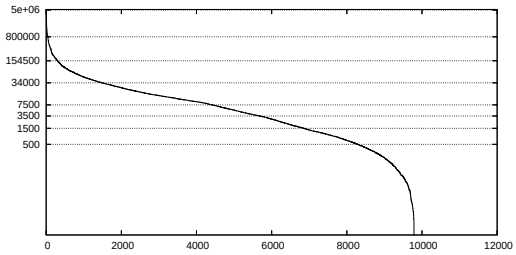

Figure 2. Package sizes for Debian 4.0. Packages are ordered by their size along the X axis, while the counts in SLOCs are represented along the Y axis (in logaritmic scale).

interesting to note that this size, around 23,000, is similar in all Debian releases, from 2.0 to 3.0, but appears to be growing since Debian 3.1, with a similar mean package size [2].

4.4 Effort and cost estimations

Using the basic COCOMO model [3], the effort to build a system with the same size as Debian 4.0 can be estimated. This estimation assumes a *classical*, proprietary development model, so it is not known if this validly estimates the amount of effort which has actually been applied to build this software. But it can give us at least an order of magnitude of the effort which would be needed in case a proprietary development model had been used.

Using the SLOC count for the Debian source packages, the data provided by the basic COCOMO model are as follows:

- Estimated effort: 881,180.28 person-months (73,431.69 person-years).
  Formula: $2.4 * KSLOC^{1.05}$
- Estimated schedule: 106.08 months (8.84 years).
  Formula: $2.5 * Effort^{0.38}$
- Estimated cost to develop: 5,358,000,000 EUR.

To get these figures, each project was estimated as though it was developed independently from the others, which in nearly all cases is true. For calculating the cost estimation, we have used the mean salary for a full-time systems programmer during 2000, according to PayScale.com and the same calculations done in [9], which is of 30,401 EURO per year, and an overhead factor of 2.4 (for an explanation on why this factor, and other details of the estimation model, see [21]).

5 Some comments and comparisons

The numbers offered in the previous section are no more than estimations. They can give us at least orders of magnitude, and allow for comparisons. But they should not be taken as exact data, there are too much sources of error and field for interpretation. In this section, we will discuss some of the more important assumptions made, and the possible sources of error. We will also compare the SLOC counts with the SLOC counts for other system, with the aim of giving the reader some context to interpret the numbers.

5.1 What is a source line of code

Since we rely on David A. Wheeler's `SLOCCount` tool for counting physical SLOC [21], we also rely on his definition for *physical source lines of code*. Therefore, we could say that we identify a SLOC when `SLOCCount` identifies a SLOC. However, `SLOCCount` has been carefully programmed to honor the usual definition for physical SLOC:

> *A physical source line of code is a line ending in a newline or end-of-file marker, and which contains at least one non-whitespace non-comment character.*

There is other similar measure, the *logical* SLOC, which sometimes is preferred. For instance, a line written in C with two semicolons would be counted as two logical SLOC, while it would be counted as one physical SLOC. However, for the purposes of this paper, the differences between both definitions of SLOC are not that important, specially when compared to other sources of error and interpretation.

5.2 Sources of inaccuracy in the SLOC counts

The counts of lines of code presented in this paper are no more than estimations. By no means do we imply that they are exact, specially when they refer to aggregates of packages. There are several factors which cause this inaccuracy, some are due to the tools used to count, some others are due to the packages' selection:

- Some files may have not been counted accurately.

  Although `SLOCCount` includes carefully designed heuristics to detect source files, and to distinguish source lines from comments, those heuristics do not always work as expected. In addition, in many cases it is difficult to distinguish automatically generated files (which should not be counted) from real files, although `SLOCCount` makes also a good effort to recognize them.

- Different perceptions in the aggregation of package families and the selection of a representative.

  As we comment in the subsection where we discuss the selection of the list of packages to count, the reasons to take a given package in or out of the list are not unquestionable. Should we count different releases of the same package? Should we count only once code present in several packages, or not? The usual criteria for measuring SLOC is "delivered source lines of code". From this point of view, all packages should be considered as they appear in the Debian release. However, this is difficult to apply when some packages are clearly evolutions (or forks) of other packages. Instead of considering all of them as "delivered", it seems more productive to consider the older ones as "beta releases" or "ancestors". However, in the libre software world it is rather common to deliver stable releases every 6 or 12 months. Those stable releases have a lot of work behind them, only to ensure stability, even if they are also the foundation for later releases.

  In most cases, we have adopted an intermediate decision: to count only once families of packages which are a line of evolution (as is the case of `emacs19` and `emacs20`), but to count separately families of packages which happen to share some code but are in themselves different developments (as is the case of `gcc` and `gnat`).

- Code reuse

  Because of the licenses being used in the libre software world there is a large quantity of code which is reused by different projects. When the code being reused is stable and properly maintained it typically is provided as a library which is packaged independently from the programs that use it. However, when developers do not agree in a common interface for a library, the shared code is copied over to projects and is consequently duplicated in different software packages.

  This also happens when a program is a fork from another program and both programs might carry independent development efforts by different upstream distributors.

  Our approach does not currently include a way to estimate or measure code reuse in the distribution. We have yet to implement mechanisms to determine how much this affects our measurements.

- Different packaging conventions

  Most packages adhere to a common packaging convention which makes packages include the original source code and Debian patches merged into the code. However, since the Debian Policy does not mandate an specific packaging convention, some packages contain upstream's code in binary format (a tar.gz) which is uncompressed and patched when the package is built to produce the binary packages.

  Also, there are a set of tools (`cdbs` and `yada` which are used in some packages to keep the original source code and maintain the patches in the `debian/` subdirectory. As above, the patches to the source code are only applied when the package is built to produce binary packages.

  Our tools currently do not handle this situation which means that packages including the source code in compressed formats will not be accounted for, and Debian patches under some circunstances will be considered maintainer scripts when they are not.

5.3 Estimation of efforts and costs

Current estimation models, and specifically COCOMO, only consider classical, proprietary development models. But libre software development models are rather different, and therefore those models may not be directly applicable. That way, we can only estimate the cost of the system, had it been developed using classical development models, but not the actual cost (in effort or in money) of the development of the software included in Debian 4.0.

Some of the differences that make it impossible to use those estimation models are:

- Continuous release process (frequent releases). The COCOMO model is based around the concept of *delivered SLOC*, which implies one point in the history of the project where the product is released. From there on, the main development effort is devoted to maintenance. On the contrary, most libre

software projects deliver releases so frequently that it could be considered as a continuous release process. This process implies the almost continuous stabilization of the code, at the same time that it evolves. Free software projects are used to improve and modify their software at the same time that they prepare it for end users.

- Bug reports and fixes. While every proprietary software system needs expensive debugging cycles which have to be fully done in-house, libre software can count on the help of people external to the project, in the form of valuable bug reports, and even fixes for them.

- Reuse, evolution, and inter-fertilization of code. Code reuse of libre software is very common in projects and is an integral part of the system being developed. It is also common that several projects develop evolutions of the same base system, in many cases with all of them using code of the others all the time. Some of this cases can also happen in proprietary developments, but even in large companies, with many open projects, they are not common, while they are the norm in libre software projects.

- Distributed development model. Although some proprietary systems are developed by geographically distributed teams, the degree of distributed development found in libre software projects is several orders of magnitude greater. There are exceptions, but usually libre software projects are carried out by people from different countries, not working for the same company, devoting different amount of effort to the project, interacting mainly through the Internet, and in most cases (specially in large projects), the developer team have never been physically together. This can effectively increase the development cost as members of a team have to interact over Internet-specific channels (mailing lists, IRC, etc.) which typically provide a lower communication bandwidth than face to face communication. Cultural differences and opinions can also generate misunderstandings in written communication which can lead to a cost increase.

Some of these factors increase the effort needed to build the software, while some others decrease it. Without analyzing in detail the impact of these (and other) factors, the estimation models in general, and COCOMO in particular, may not be directly applicable to libre software development.

5.4 Comparison with size estimations of other systems

To put the numbers shown above into context, here we offer estimations for the size of some operating systems. Note that these comparisons are always difficult, and specially when we compare with proprietary systems.

As reported in [15] (for Windows 2000), in [21] and [12] (for Red Hat Linux), some additional studies by us not yet published (for Fedora Core 3) and in [19] and [16] (for the rest of the systems), this is the estimated size for several operating systems, in lines of code (all numbers are just approximations):

- Microsoft Windows 3.1: 3,000,000
- Sun Solaris: 7,500,000
- Microsoft Windows 95: 15,000,000
- Red Hat Linux 6.2: 17,000,000
- Microsoft Windows 2000: 29,000,000
- Red Hat Linux 7.1: 30,000,000
- Microsoft XP: 40,000,000
- Red Hat Linux 8.0: 50,000,000
- Fedora Core 3: 70,000,000
- Debian 3.0: 105,000,000
- Debian 3.1: 245,000,000
- Debian 4.0: 283,000,000

Most of these estimations (in fact, all of them, except for Red Hat Linux) are not detailed, and is difficult to know what they consider as a line of code. However, the estimations should be similar enough to SLOC counting methods to be suitable for comparison.

Note also that, while both the Red Hat's and Debian's operating systems include many applications for a wide range of purposes, with even several applications in the same category in many situations, both Microsoft's and Sun's operating systems are much more limited in this way, with some software packages sold outside of the core operating system even if developed by the same company. If the size of the most common applications used in those environments (for example, office suites) were added, their size would be much larger. However,

it is also true that all those applications are not developed nor put together by the same team of developers, as is the case in Linux-based distributions.

From these numbers, it can be seen that Linux-based distributions in general, and Debian releases in particular, are some of the largest pieces of software ever put together by a group of developers.

6 Conclusions

In this paper, we have presented some results of our work on counting the number of SLOC of Debian 4.0. They represent the state of the Debian GNU/Linux distribution in April 2007. Our estimation is that it amounts for more than 283,000,000 SLOC. Using the COCOMO model, this implies a cost (using traditional, proprietary software development models) close to 5,400 million Euros and an effort of more than 73,000 person-years. The list of the largest packages, and an analysis by programming language used are also provided.

When coming to the details, some interesting data can be shown. For instance, the most popular language in the distribution is C (close to 51%), followed by C++ (more than 18%), Shell (about 10%), Java, PERL and LISP (both about 3%) and other. The largest packages in Debian 4.0 are OpenOffice.org (nearly 5,200,000 SLOC), the Linux Kernel (about 5,000,000 SLOC), ia32-libs (nearly 4,000,000), and gcc-4.1 (about 3,600,000).

There are not many detailed studies of the size of modern, complete operating systems. Of them, the work by David A. Wheeler, counting the size of Red Hat 6.2 and Red Hat 7.1 is the most comparable. Another interesting paper, with some intersection with this paper is [10], a study on the evolution over time of the Linux kernel. Some other papers, already referenced, provide total counts of some Sun and Microsoft operating systems, but they are not detailed enough, except for providing estimations for the whole of the system.

Finally, we find it important to repeat once more that we are offering only estimations. However, we believe they are accurate enough to draw some conclusions and to compare with other systems.

References

[1] J. J. Amor, J. M. Gonzalez-Barahona, G. Robles, and I. Herraiz. Measuring libre software using Debian 3.1 (Sarge) as a case study: preliminary results. *Upgrade Magazine*, Aug. 2005.

[2] J. J. Amor, G. Robles, J. M. González-Barahona, and I. Herraiz. From pigs to stripes: A travel through Debian. In *Proceedings of the DebConf5 (Debian Annual Developers Meeting)*, Helsinki, Finland, July 2005.

[3] B. B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[4] Canonical. Ubuntu project, Apr. 2004. `http://www.ubuntu.com/`.

[5] Debian. Debian free software guidelines (part of the debian social contract), Apr. 2004. `http://www.debian.org/social_contract`.

[6] Debian. Debian social contract, Apr. 2004. `http://www.debian.org/social_contract`.

[7] Debian. Debian policy manual, Oct. 2006. `http://www.debian.org/doc/debian-policy/`.

[8] Debian. Debian gnu/linux 4.0 release information, Apr. 2007. `http://www.debian.org/releases/4.0`.

[9] R. A. Ghosh et al. Study on the: Economic impact of open source software on innovation and the competitiveness of the information and communication technologies (ict) sector in the eu. Technical report, UNU-MERIT, Nov. 2006. `http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf`.

[10] M. W. Godfrey and Q. Tu. Evolution in Open Source software: A case study. In *Proceedings of the International Conference on Software Maintenance*, pages 131–142, San Jose, California, 2000.

[11] J. M. Gonzalez-Barahona, M. A. Ortuno Perez, P. de las Heras, J. Centeno, and V. Matellan. Counting potatoes: the size of Debian 2.2. *Upgrade Magazine*, II(6):60–66, Dec. 2001.

[12] J. M. González-Barahona, G. Robles, M. Ortuño-Pérez, L. Rodero-Merino, J. Centeno-González, V. Matellán-Olivera, E. Castro-Barbero, and P. de-las Heras-Quirós. Anatomy of two gnu/linux distributions, 2004. Chapter in book "Free/Open Source Software Development" edited by Stefan Koch and published by Idea Group, Inc.

[13] Hispalinux. Proyecto metadistros, Feb. 2004. `https://forja.rediris.es/projects/metadistros/`.

[14] K. Knopper. Knoppix: Live linux filesystem on cd, Jan. 2003. `http://www.knopper.net/knoppix/index-en.html`.

[15] M. Lucovsky. From nt os/2 to windows 2000 and beyond - a software-engineering odyssey. 4th usenix windows systems symposium, 2000. `http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky_html/`.

[16] G. McGraw. From the ground up: the dimacs software security workshop. *IEEE Security and Privacy*, 1(2):59–66, 2003.

[17] G. Robles, J. J. Amor, and J. M. González-Barahona. Debian counting, Apr. 2007. http://debian-counting.libresoft.es/.

[18] G. Robles, J. M. González-Barahona, and M. Michlmayr. Evolution of volunteer participation in libre software projects: evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems*, pages 100–107, Genoa, Italy, July 2005.

[19] B. Scheneier. Software complexity and security. *Crypto-Gram Newsletter*, 2000.

[20] D. Wheeler. Sloccount. http://www.dwheeler.com/sloccount/.

[21] D. A. Wheeler. More than a gigabuck: Estimating GNU/Linux's size, June 2001. http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html.