More Than a Gigabuck: Estimating GNU/Linux's Size

David A. Wheeler (dwheeler@dwheeler.com)

June 30, 2001 (updated July 29, 2002)
Version 1.07

This paper analyzes the amount of source code in GNU/Linux, using Red Hat Linux 7.1 as a representative GNU/Linux distribution, and presents what I believe are interesting results.

In particular, it would cost over $1 billion ($1,000 million - a Gigabuck) to develop this GNU/Linux distribution by conventional proprietary means in the U.S. (in year 2000 U.S. dollars). Compare this to the $600 million estimate for Red Hat Linux version 6.2 (which had been released about one year earlier). Also, Red Hat Linux 7.1 includes over 30 million physical source lines of code (SLOC), compared to well over 17 million SLOC in version 6.2. Using the COCOMO cost model, this system is estimated to have required about 8,000 person-years of development time (as compared to 4,500 person-years to develop version 6.2). Thus, Red Hat Linux 7.1 represents over a 60% increase in size, effort, and traditional development costs over Red Hat Linux 6.2. This is due to an increased number of mature and maturing open source / free software programs available worldwide.

Many other interesting statistics emerge. The largest components (in order) were the Linux kernel (including device drivers), Mozilla (Netscape's open source web system including a web browser, email client, and HTML editor), the X Window system (the infrastructure for the graphical user interface), gcc (a compilation system), gdb (for debugging), basic binary tools, emacs (a text editor and far more), LAPACK (a large Fortran library for numerical linear algebra), the Gimp (a bitmapped graphics editor), and MySQL (a relational database system). Note that some projects (in particular KDE and GNOME) are in aggregate large enough to be one of the largest components, but because they are developed and distributed as a large number of smaller components, their totals don't appear in the list of largest components. The languages used, sorted by the most lines of code, were C (71% - was 81%), C++ (15% - was 8%), shell (including ksh), Lisp, assembly, Perl, Fortran, Python, tcl, Java, yacc/bison, expect, lex/flex, awk, Objective-C, Ada, C shell, Pascal, and sed.

The predominant software license is the GNU GPL. Slightly over half of the software is simply licensed using the GPL, and the software packages using the copylefting licenses (the GPL and LGPL), at least in part or as an alternative, accounted for 63% of the code. In all ways, the copylefting licenses (GPL and LGPL) are the dominant licenses in this GNU/Linux distribution. In contrast, only 0.2% of the software is public domain.

This paper is an update of my previous paper on estimating GNU/Linux's size, which measured Red Hat Linux 6.2 [Wheeler 2001]. Since Red Hat Linux 6.2 was released in March 2000, and Red Hat Linux 7.1 was released in April 2001, this paper shows what's changed over approximately one year. More information is available at http://www.dwheeler.com/sloc.

1. Introduction

The GNU/Linux operating system has gone from an unknown to a powerful market force. Netcraft found that, of the systems running web servers on June 2001, GNU/Linux was now the second most popular operating system (with 29.6%, versus Windows' 49.6%) [Netcraft 2001]. Another survey, of primarily European and educational sites, found that GNU/Linux was used more than any other operating system (of the sites it surveyed) [Zoebelein 1999]. IDC found that 25% of all server operating systems purchased in 1999 were GNU/Linux, making it second only to Windows NT's 38% [Shankland 2000a].

There appear to be many reasons for this, and not simply because GNU/Linux can be obtained at no or low cost. For example, experiments suggest that GNU/Linux is highly reliable. A 1995 study of a set of individual components found that the GNU and GNU/Linux components had a significantly higher reliability than their proprietary Unix competitors (6% to 9% failure rate with GNU and Linux, versus an average 23% failure rate with the proprietary software using their measurement technique) [Miller 1995]. A ten-month experiment in 1999 by ZDnet found that, while Microsoft's Windows NT crashed every six weeks under a ``typical'' intranet load, using the same load and request set the GNU/Linux systems (from two different distributors) *never* crashed [Vaughan-Nichols 1999].

However, possibly the most important reason for GNU/Linux's popularity among many developers and users is that its source code is generally ``open source software'' and/or ``free software''. A program that is ``open source software'' or ``free software'' is essentially a program whose source code can be obtained, viewed, changed, and redistributed without royalties or other limitations of these actions. A more formal definition of ``open source software'' is available from the Open Source Initiative [OSI 1999], a more formal definition of ``free software'' (as the term is used in this paper) is available from the Free Software Foundation [FSF 2000], and other general information about these topics is available at Wheeler [2000a]. Quantitative rationales for using open source / free software is given in Wheeler [2000b]. The GNU/Linux operating system is actually a suite of components, including the Linux kernel on which it is based, and it is packaged, sold, and supported by a variety of distributors. The Linux kernel is ``open source software''/``free software'', and this is also true for all (or nearly all) other components of a typical GNU/Linux distribution. Open source software/free software frees users from being captives of a particular vendor, since it permits users to fix any problems immediately, tailor their system, and analyze their software in arbitrary ways.

Surprisingly, although anyone can analyze GNU/Linux for arbitrary properties, I have found little published analysis of the amount of source lines of code (SLOC) contained in a GNU/Linux distribution. Microsoft unintentionally published some analysis data in the documents usually called ``Halloween I" and ``Halloween II" [Halloween I] [Halloween II]. Another study focused on the Linux kernel and its growth over time is by Godfrey [2000]; this is an interesting study but it focuses solely on the Linux kernel (not the entire operating system). Paul G. Allen posted some results from running Scientific Toolworks, Inc.'s tools on the Linux kernel, but this analysis only considered C code (including headers) - ignoring the many other languages used in constructing the Linux kernel (e.g., assembly language), and only concentrating on the kernel. The Free Code Graphing Project at http://fcgp.sourceforge.net generates a graphical representation of a program (currently, the Linux kernel), but only of the C code. In a previous paper, I examined Red Hat Linux 6.2 and the numbers from the Halloween papers [Wheeler 2001].

This paper updates my previous paper, showing estimates of the size of one of today's GNU/Linux distributions, and it estimates how much it would cost to rebuild this typical GNU/Linux distribution using traditional software development techniques. Various definitions and assumptions are included, so that others can understand exactly what these numbers mean. I have intentionally written this paper so that you do *not* need to read the previous version of this paper first.

For my purposes, I have selected as my ``representative" GNU/Linux distribution Red Hat Linux version 7.1. I believe this distribution is reasonably representative for several reasons:

1. Red Hat Linux is the most popular Linux distribution sold in 1999 according to IDC [Shankland 2000b]. Red Hat sold 48% of all copies in 1999; the next largest distribution in market share sales was SuSE (a German distributor) at 15%. Not all GNU/Linux copies are ``sold" in a way that this study would count, but the study at least shows that Red Hat's distribution is a popular one.
2. Many distributions (such as Mandrake) are based on, or were originally developed from, a version of Red Hat Linux. This doesn't mean the other distributions are less capable, but it suggests that these other distributions are likely to have a similar set of components.
3. All major general-purpose distributions support (at least) the kind of functionality supported by Red Hat Linux, if for no other reason than to compete with Red Hat.
4. All distributors start with the same set of open source software projects from which to choose components to integrate. Therefore, other distributions are likely to choose the same components or similar kinds of components with often similar size for the same kind of functionality.

Different distributions and versions would produce different size figures, but I hope that this paper will be enlightening even though it doesn't try to evaluate ``all" distributions. Note that some distributions (such as SuSE) may decide to add many more applications, but also note this would only create larger (not smaller) sizes and estimated levels of effort. At the time that I began this project, version 7.1 was the latest version of Red Hat Linux available, so I selected that version for analysis.

Note that Red Hat Linux 6.2 was released on March 2000, Red Hat Linux 7 was released on September 2000 (I have not counted its code), and Red Hat Linux 7.1 was released on April 2001. Thus, the differences between Red Hat Linux 7.1 and 6.2 show differences accrued over 13 months (approximately one year).

Clearly there is far more open source / free software available worldwide than is counted in this paper. However, the job of a distributor is to examine these various options and select software that they believe is both sufficiently mature and useful to their target market. Thus, examining a particular distribution results in a selective analysis of such software.

Section 2 briefly describes the approach used to estimate the ``size'' of this distribution (more details are in Appendix A). Section 3 discusses some of the results. Section 4 presents conclusions, followed by an appendix. GNU/Linux is often called simply ``Linux'', but technically Linux is only the name of the operating system kernel; to eliminate ambiguity this paper uses the term ``GNU/Linux'' as the general name for the whole system and ``Linux kernel'' for just this inner kernel.

2. Approach

My basic approach was to:

1. install the source code files in uncompressed format; this requires carefully selecting the source code to be analyzed.
2. count the number of source lines of code (SLOC); this requires a careful definition of SLOC.
3. use an estimation model to estimate the effort and cost of developing the same system in a proprietary manner; this requires an estimation model.
4. determine the software licenses of each component and develop statistics based on these categories.

More detail on this approach is described in Appendix A. A few summary points are worth mentioning here, however.

2.1 Selecting Source Code

I included all software provided in the Red Hat distribution, but note that Red Hat no longer includes software packages that only apply to other CPU architectures (and thus packages not applying to the x86 family were excluded). I did not include ``old'' versions of software, or ``beta'' software where non-beta was available. I did include ``beta'' software where there was no alternative, because some developers don't remove the ``beta'' label even when it's widely used and perceived to be reliable.

I used md5 checksums to identify and ignore duplicate files, so if the same file contents appeared in more than one file, it was only counted once (as a tie-breaker, such files are assigned to the first build package it applies to in alphabetic order).

The code in makefiles and Red Hat Package Manager (RPM) specifications was not included. Various heuristics were used to detect automatically generated code, and any such code was also excluded from the count. A number of other heuristics were used to determine if a language was a source program file, and if so, what its language was.

Since different languages have different syntaxes, I could only measure the SLOC for the languages that my tool (sloccount) could detect and handle. The languages sloccount could detect and handle are Ada, Assembly, awk, Bourne shell and variants, C, C++, C shell, Expect, Fortran, Java, lex/flex, LISP/Scheme, Makefile, Objective-C, Pascal, Perl, Python, sed, SQL, TCL, and Yacc/bison. Other languages are not counted; these include XUL (used in Mozilla), Javascript (also in Mozilla), PHP, and Objective Caml (an OO dialect of ML). Also code embedded in data is not counted (e.g., code embedded in HTML files). Some systems use their own built-in languages; in general code in these languages is not counted.

2.2 Defining SLOC

The ``physical source lines of code'' (physical SLOC) measure was used as the primary measure of SLOC in this paper. Less formally, a physical SLOC in this paper is a line with something other than comments and whitespace (tabs and spaces). More specifically, physical SLOC is defined as follows: ``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.'' Comment delimiters (characters other than newlines starting and ending a comment) were considered comment characters. Data lines only including whitespace (e.g., lines with only tabs and spaces in multiline strings) were not included.

Note that the ``logical'' SLOC is not the primary measure used here; one example of a logical SLOC measure would be the ``count of all terminating semicolons in a C file.'' The ``physical'' SLOC was chosen instead of the ``logical'' SLOC because there were so many different languages that needed to be measured. I had trouble getting freely-available tools to work on this scale, and the non-free tools were too expensive for my budget (nor is it certain that they would have fared any better). Since I had to develop my own tools, I chose a measure that is much easier to implement. Park [1992] actually recommends the use of the physical SLOC measure (as a minimum), for this and other reasons. There are disadvantages to the ``physical'' SLOC measure. In particular, physical SLOC measures are sensitive to how the code is formatted. However, logical SLOC measures have problems too. First, as noted, implementing tools to measure logical SLOC is more difficult, requiring more sophisticated analysis of the code. Also, there are many different possible logical SLOC measures, requiring even more careful definition. Finally, a logical SLOC measure must be redefined for every language being measured, making inter-language comparisons more difficult. For more information on measuring software size, including the issues and decisions that must be made, see Kalb [1990], Kalb [1996], and Park [1992].

Note that this required that every file be categorized by language type (so that the correct syntax for comments, strings, and so on could be applied). Also, automatically generated files had to be detected and ignored. Thankfully, my tool ``sloccount'' does this automatically.

2.3 Estimation Models

This decision to use physical SLOC also implied that for an effort estimator I needed to use the original COCOMO cost and effort estimation model (see Boehm [1981]), rather than the newer ``COCOMO II'' model. This is simply because COCOMO II requires logical SLOC as an input instead of physical SLOC.

Basic COCOMO is designed to estimate the time from product design (after plans and requirements have been developed) through detailed design, code, unit test, and integration testing. Note that plans and requirement development are not included. COCOMO is designed to include management overhead and the creation of documentation (e.g., user manuals) as well as the code itself. Again, see Boehm [1981] for a more detailed description of the model's assumptions. Of particular note, basic COCOMO does not include the time to develop translations to other human languages (of documentation, data, and program messages) nor fonts.

There is reason to believe that these models, while imperfect, are still valid for estimating effort in open source / free software projects. Although many open source programs don't need management of human resources, they still require technical management, infrastructure maintenance, and so on. Design documentation is captured less formally in open source projects, but it's often captured by necessity because open source projects tend to have many developers separated geographically. Clearly, the systems must still be programmed. Testing is still done, although as with many of today's proprietary programs, a good deal of testing is done through alpha and beta releases. In addition, quality is enhanced in many open source projects through peer review of submitted code. The estimates may be lower than the actual values because they don't include estimates of human language translations and fonts.

Each software source code package, once uncompressed, produced zero or more ``build directories'' of source code. Some packages do not actually contain source code (e.g., they only contain configuration information), and some packages are collections of multiple separate pieces (each in different build directories), but in most cases each package uncompresses into a single build directory containing the source code for that package. Each build directory had its effort estimation computed separately; the efforts of each were then totalled. This approach assumes that each build directory was developed essentially separately from the others, which in nearly all cases is quite accurate. This approach slightly underestimates the actual effort in the rare cases where the development of the code in separate build directories are actually highly interrelated; this effect is not expected to invalidate the overall results.

For programmer salary averages, I used a salary survey from the September 4, 2000 issue of ComputerWorld; their survey claimed that this annual programmer salary averaged $56,286 in the United States. I was unable to find a publicly-backed average value for overhead, also called the ``wrap rate.'' This value is necessary to estimate the costs of office space, equipment, overhead staff, and so on. I talked to two cost analysts, who suggested that 2.4 would be a reasonable overhead (wrap) rate. Some Defense Systems Management College (DSMC) training material gives examples of 2.3 (125.95%+100%) not including general and administrative (G&A) overhead, and 2.81 when including G&A (125% engineering overhead, plus 25% on top of that amount for G&A) [DSMC]. This at least suggests that 2.4 is a plausible estimate. Clearly, these values vary widely by company and region; the information provided in this paper is enough to use different numbers if desired. These are the same values as used in my last report.

2.4 Determining Software Licenses

A software license determines how that software can be used and reused, and open source software licensing has been a subject of great debate. The Software Release Practice HOWTO [Raymond 2001] discusses briefly why license choices are so important to open source / free software projects:

> *The license you choose defines the social contract you wish to set up among your co-developers and users ...*
>
> *Who counts as an author can be very complicated, especially for software that has been worked on by many hands. This is why licenses are important. By setting out the terms under which material can be used, they grant rights to the users that protect them from arbitrary actions by the copyright holders.*
>
> *In proprietary software, the license terms are designed to protect the copyright. They're a way of granting a few rights to users while reserving as much legal territory is possible for the owner (the copyright holder). The copyright holder is very important, and the license logic so restrictive that the exact technicalities of the license terms are usually unimportant.*
>
> *In open-source software, the situation is usually the exact opposite; the copyright exists to protect the license. The only rights the copyright holder always keeps are to enforce the license. Otherwise, only a few rights are reserved and most*

choices pass to the user. In particular, the copyright holder cannot change the terms on a copy you already have. Therefore, in open-source software the copyright holder is almost irrelevant -- but the license terms are very important.

Well-known open source licenses include the GNU General Public License (GPL), the GNU Library/Lesser General Public License (LGPL), the MIT (X) license, the BSD license, and the Artistic license. The GPL and LGPL are termed ``copylefting'' licenses, that is, the license is designed to prevent the code from becoming proprietary. See Perens [1999] for more information comparing these licenses. Obvious questions include ``what license(s) are developers choosing when they release their software'' and ``how much code has been released under the various licenses?''

An approximation of the amount of software using various licenses can be found for this particular distribution. Red Hat Linux uses the Red Hat Package Manager (RPM), and RPM supports capturing license data for each package (these are the ``Copyright'' and ``License'' fields in the specification file). I used this information to determine how much code was covered by each license. Since this field is simply a string of text, there were some variances in the data that I had to clean up, for example, some entries said ``GNU'' while most said ``GPL''. In some cases Red Hat did not include licensing information with a package. In that case, I wrote a program to attempt to determine the license by looking for certain conventional filenames and contents.

This is an imperfect approach. Some packages contain different pieces of code with difference licenses applying to different pieces. Some packages are ``dual licensed'', that is, they are released under more than one license. Sometimes these other licenses are noted, while at other times they aren't. There are actually two BSD licenses (the ``old'' and ``new'' licenses), but the specification files don't distinguish between them. Also, if the license wasn't one of a small set of common licenses, Red Hat tended to assigned nondescriptive phrases such as ``distributable''. My automated techniques were limited too, in particular, while some licenses (e.g., the GPL and LGPL) are easy to recognize automatically, BSD-like and MIT-like licenses vary the license text and so are more difficult to recognize automatically (and some changes to the license would render them non-open source, non-free software). Thus, when Red Hat did not identify a package's license, a program dual licensed under both the BSD and GPL license might only be labelled as having the GPL using these techniques. Nevertheless, this approach is sufficient to give some insight into the amount of software using various licenses. Future research could examine each license in turn and categorize them; such research might require several lawyers to determine when two licenses in certain circumstances are ``equal.''

One program worth mentioning in this context is Python, which has had several different licenses. Version 1.6 and later (through 2.1) had more complex licenses that the Free Software Foundation (FSF) believes were incompatible with the GPL. Recently this was resolved by another change to the Python license to make Python fully compatible with the GPL. Red Hat Linux 7.1 includes an older version of Python (1.5.2), presumably because of these licensing issues. It can't be because Red Hat is unaware of later versions of Python; Red Hat uses Python in its installation program (which it developed and maintains). Hopefully, the recent resolution of license incompatibilities with the GPL license will enable Red Hat to include the latest versions of Python in the future. In any case, there are several different Python-specific licenses, all of which can legitimately be called the ``Python'' license. Red Hat has labelled Python itself as having a ``Distributable'' license, and package Distutils-1.0.1 is labelled with the ``Python'' license; these labels are kept in this paper.

3. Results

Given this approach, here are some of the results. Section 3.1 presents the largest components (sorted by SLOC), section 3.2 presents results specifically from the Linux kernel's SLOC, section 3.3 presents total counts by language, section 3.4 presents total counts of files (instead of SLOC), section 3.5 presents total counts grouped by their software licenses, section 3.6 presents total SLOC counts, and section 3.7 presents effort and cost estimates.

3.1 Largest Components by SLOC

Here are the top 35 largest components (as measured by number of source lines of code), along with their licenses (see section 2.4 for how these license values were determined). In the language section, ``ansic'' means C code, ``asm'' is assembly, ``sh'' is Bourne shell and related shells, and ``cpp'' is C++.

```
SLOC    Directory       SLOC-by-Language (Sorted)
2437470 kernel-2.4.2    ansic=2285657,asm=144411,sh=3035,perl=2022,yacc=1147,
                        tcl=576,lex=302,awk=248,sed=72
                        [GPL]
2065224 mozilla         cpp=1279902,ansic=739470,perl=21220,sh=13717,asm=5212,
                        java=3107,yacc=1831,lex=470,csh=271,sed=24
                        [MPL]
1837608 XFree86-4.0.3   ansic=1750460,asm=35397,cpp=20725,sh=14666,tcl=9182,
                        yacc=3360,perl=1675,lex=1608,awk=393,csh=85,sed=57
                        [MIT]
984076  gcc-2.96-20000731 ansic=789901,cpp=126738,yacc=19272,sh=17993,asm=14559,
                        lisp=7161,fortran=3814,exp=3705,objc=479,sed=310,perl=144
                        [GPL]
967263  gdb+dejagnu-20010316 ansic=871288,exp=58422,sh=12054,cpp=8252,yacc=5906,
                        asm=5031,tcl=4477,lisp=1403,sed=248,awk=170,java=7,fortran=5
                        [GPL]
690983  binutils-2.10.91.0.2 ansic=489993,asm=161236,exp=13234,sh=12835,
                        yacc=5665,cpp=4777,lex=1488,perl=776,sed=561,lisp=394,awk=24
                        [GPL]
646692  glibc-2.2.2     ansic=548722,asm=88413,sh=6036,perl=2120,awk=1037,
                        yacc=315,sed=49
                        [LGPL]
627626  emacs-20.7      lisp=453898,ansic=169956,sh=2622,perl=884,asm=253,
                        csh=9,sed=4
                        [GPL]
474829  LAPACK          fortran=473590,ansic=1239
                        [Freely distributable]
455980  gimp-1.2.1      ansic=427967,perl=17482,lisp=9648,yacc=502,sh=381
                        [GPL, LGPL]
402799  mysql-3.23.36   ansic=249350,cpp=84068,perl=25088,tcl=18980,sh=18323,
                        asm=3987,awk=1436,java=1149,sed=418
                        [LGPL]
395194  tcltk-8.3.1     ansic=291457,tcl=84322,sh=12259,exp=5742,yacc=876,
                        awk=273,perl=265
                        [BSD]
345949  kdebase-2.1.1   cpp=181210,ansic=158682,sh=4880,perl=1155,python=22
                        [GPL]
323730  Mesa-3.4        ansic=286437,cpp=18189,asm=10002,sh=7611,objc=1184,
                        python=307
                        [GPL/MIT]
321123  perl-5.6.0      perl=146755,ansic=118233,sh=49377,lisp=5739,yacc=996,
                        java=23
                        [Artistic or GPL]
318430  libgcj          ansic=191432,cpp=56843,java=41716,sh=15581,asm=11262,
                        exp=841,perl=731,awk=24
                        [GPL]
304819  teTeX-1.0       ansic=223491,perl=49789,sh=17634,cpp=9407,pascal=1546,
                        yacc=1507,awk=622,lex=323,sed=314,asm=139,csh=47
                        [Distributable]
298742  qt-2.3.0        cpp=259310,ansic=34578,yacc=2444,sh=1493,lex=480,
                        perl=422,lisp=15
                        [GPL]
286113  postgresql-7.0.3 ansic=237184,java=17540,yacc=9740,sh=8975,tcl=7751,
                        lex=1810,perl=1276,python=959,cpp=801,asm=70,csh=5,sed=2
                        [BSD]
283785  kdelibs-2.1.1   cpp=261334,ansic=17578,sh=1887,java=1538,perl=731,
                        yacc=607,lex=110
                        [LGPL]
277502  xemacs-21.1.14  ansic=199927,lisp=73366,sh=2948,perl=930,asm=247,
                        csh=62,sed=22
                        [GPL]
264528  gs5.50          ansic=259471,cpp=2266,asm=968,sh=823,lisp=405,perl=336,
                        yacc=201,lex=58
                        [GPL]
227354  krb5-1.2.2      ansic=197886,exp=19124,sh=5140,yacc=2474,perl=1529,
                        awk=393,python=348,lex=190,csh=147,sed=123
                        [MIT]
215473  vnc_unixsrc     ansic=212766,cpp=848,asm=780,perl=648,sh=431
                        [GPL]
213818  koffice-2.0.1   cpp=197637,sh=7296,yacc=3791,ansic=3213,perl=1801,
                        lex=80
                        [GPL]
202842  openssl-0.9.6   ansic=131874,cpp=25744,perl=14737,asm=12428,python=10171,
                        yacc=3297,sh=2641,tcl=1583,lisp=224,objc=143
                        [BSD-like]
200908  Python-1.5.2    python=101017,ansic=96521,lisp=2353,sh=673,perl=342,
                        sed=2
                        [Distributable]
194799  bind-9.1.0      ansic=173830,sh=12101,yacc=6025,perl=2830,tcl=13
                        [BSD-like]
192394  xpdf-0.92       cpp=167135,ansic=21621,sh=3638
                        [GPL]
191379  php-4.0.4pl1    ansic=173334,cpp=7033,sh=6591,lex=1867,yacc=1569,
                        java=437,awk=367,perl=181
                        [PHP]
190950  pine4.33        ansic=190020,sh=838,csh=62,perl=30
                        [Freely distributable]
173492  abi             cpp=159595,ansic=12605,perl=725,sh=550,python=17
                        [GPL]
167663  kdemultimedia-2.1.1 cpp=140731,ansic=23844,tcl=1004,sh=800,asm=598,
                        lex=578,perl=106,awk=2
                        [GPL]
163449  4Suite-0.10.1   python=91445,ansic=72004
                        [Apache-like]
159301  linuxconf-1.24r2 cpp=142970,perl=6738,sh=3821,java=3074,ansic=2613,
                        python=85
                        [GPL]
```

Note that the operating system kernel (Linux) is the largest single component, at over 2.4 million lines of code (mostly in C); that compares to 1.5 million lines of code in Red Hat 6.2. See section 3.2 for a more detailed discussion about the Linux kernel.

The next largest component is Mozilla; this is large because it's really a suite of applications including a web browser, email reader, news reader, HTML editor, and so on. Mozilla is the basis for Netscape Navigator 6.0. Mozilla was not included at all in Red Hat Linux 6.2.

The next largest component is the X Window system, a critical part of the graphical user interface (GUI). Given the importance of GUIs, the long history of this program (giving it time to gain functionality and size), and the many incompatible video displays it must support, this is perhaps not surprising.

Next is the gcc compilation system, including the C and C++ compilers, the symbolic debugger, a set of utilities for binary files, and the C library (which is actually used by most other language libraries as well). Emacs is next largest, which should not be a real surprise; some users use nothing but emacs (e.g., reading their email via emacs), using emacs as a kind of virtual operating system.

Note that language implementations tend to be written in themselves, particularly for their libraries. Perl's implementation is written mostly in Perl, and Python is written mostly in Python. Intriguingly, this is not true for Tcl.

In many senses, what is the ``largest'' component is an artifact of packaging. GNOME and KDE are actually huge, but both are packaged as a set of components instead of being delivered as a single large component. The amount of C code in ``kdebase'' seemed suspiciously high to one KDE developer, but it turns out that Red Hat includes ``lesstiflite'' (a Motif clone) in kdebase to support Netscape plug-ins. My thanks to Waldo Bastian (of KDE) for pointing out this unusual situation in kdebase and determining its cause, and to Bernhard Rosenkraenzer (of Red Hat) for confirming the reason for this kdebase addition. Éric Bischoff noted that KDE includes all the files beginning with kde, plus koffice (and in the latest versions of KDE, arts); using this definition and totalling the numbers in the summary table, KDE (with common applications) has 1,693,235 SLOC. In a similar manner, by totalling all the packages with ``gnome'' in them, as well as gnumeric, abi (abiword), and the Gimp, GNOME (with common applications) has 1,195,992 SLOC. One trouble here is determining exactly what to count; for example, do you include common applications, and if so, which ones? What about the graphical toolkit? For these reasons, I've simply let the component list stand.

For a complete list of all components and their SLOC counts, see http://www.dwheeler.com/sloc/redhat71-v1/summary.

3.2 Examination of the Linux Kernel's SLOC

Since the largest single component was the Linux kernel (at over 2.4 million SLOC), I examined it further, to learn why it was so large and determine its ramifications.

I found that over 1,400,000 lines (57% of the Linux kernel) was in the ``drivers'' subdirectory, thus, the primary reason the kernel is so large is that it supports so many different kinds of peripherals. No other subdirectory comes close to this size - the second largest is the ``arch'' directory (at over 446,000 SLOC, 18% of the kernel), which contains the architecture-unique code for each CPU architecture. Supporting many different filesystems also increases its size, but not as much as expected; the entire filesystem code is over 168,000 SLOC.

Richard Stallman and others have argued that the resulting system often called ``Linux'' should instead be called ``GNU/Linux'' [Stallman 2000]. In particular, by hiding GNU's contributions (through not including GNU's name), many people are kept unaware of the GNU project and its purpose, which is to encourage a transition to ``free software'' (free as in the freedom to use, modify, and redistribute software for any purpose). Certainly, the resulting system was the intentional goal and result of the GNU project's efforts. Another argument used to justify the term ``GNU/Linux'' is that it is confusing if both the entire operating system and the operating system kernel are both called ``Linux''. Using the term ``Linux'' is particularly bizarre for GNU/Hurd, which takes the Debian GNU/Linux distribution and swaps out one component: the Linux kernel.

The data here can be used to justify calling the system either ``Linux'' or ``GNU/Linux.'' It's clear that the largest single component in the operating system is the Linux kernel, so it's at least understandable how so many people have chosen to name the entire system after its largest single component (``Linux''). It's also clear that there are many contributors, not just the GNU project itself, and some of those contributors do not agree with the GNU project's philosophy. On the other hand, many of the largest components of the system are essentially GNU projects: gcc, gdb, emacs, binutils (a set of commands for binary files), and glibc (the C library). Other GNU projects in the system include binutils, bash, gawk, make, textutils, sh-utils, gettext, readline, automake, tar, less, findutils, diffutils, and grep. This is not even counting GNOME, a GNU project. In short, the total of the GNU project's code is *much* larger than the Linux kernel's size. Thus, by comparing the total contributed effort, it's certainly justifiable to call the entire system ``GNU/Linux'' and not just ``Linux,'' and using the term GNU/Linux both credits its contributions and eliminates some ambiguity. Thus, I've decided to switch to the ``GNU/Linux'' terminology here.

For more information on the sizes of the Linux kernel components, see http://www.dwheeler.com/sloc/redhat71-v1/kernel_sloc.

3.3 Total Counts by Language

Here are the various programming languages, sorted by the total number of source lines of code (using the naming conventions of sloccount, the program used to count SLOC):

Language	**SLOC (%)**
C	21461450 (71.18%)
C++	4575907 (15.18%)
Shell (Bourne-like)	793238 (2.63%)
Lisp	722430 (2.40%)
Assembly	565536 (1.88%)
Perl	562900 (1.87%)
Fortran	493297 (1.64%)
Python	285050 (0.95%)
Tcl	213014 (0.71%)
Java	147285 (0.49%)
yacc/bison	122325 (0.41%)
Expect	103701 (0.34%)
lex/flex	41967 (0.14%)
awk/gawk	17431 (0.06%)
Objective-C	14645 (0.05%)
Ada	13200 (0.04%)
C shell	10753 (0.04%)
Pascal	4045 (0.01%)
sed	3940 (0.01%)

Here you can see that C is pre-eminent (with over 71% of the code), followed by C++, shell, LISP, assembly, Perl, Fortran, and Python. Some of the languages with smaller counts (such as objective-C and Ada) show up primarily as test cases or bindings to support users of those languages. Nevertheless, it's nice to see at least some support for a variety of languages, since each language has some strength for some type of application.

C++ has about 4.5 million lines of code, a very respectable showing, but is far less than C (over 21 million SLOC). Still, there's increasing use of C++ code; in the last survey, C had 80.55% and C++ had 7.51%. There is slightly less C code in the total percentage of code, most of which is being taken by C++. One could ask why there's so much more C code, particularly against C++. One possible argument is that well-written C++ takes fewer lines of code than does C; while this is often true, that's unlikely to entirely explain this. Another important factor is that many of the larger programs were written before C++ became widely used, and no one wishes to rewrite their C programs into C++. Also, there are a significant number of software developers who prefer C over C++ (e.g., due to simplicity of understanding the entire language), which would certainly affect these numbers. There have been several efforts in the past to switch from C to C++ in the Linux kernel, and they have all failed (for a variety of reasons).

LISP continues to place very highly, far more than Perl, Python, Fortran, or Java. LISP is used in many components, but its high placement is due to the widespread use of emacs. Emacs itself is written in primarily in its own variant of LISP, and the emacs package itself accounts for 87% (627626/722430) of the LISP code. In addition, many languages include sophisticated (and large) emacs modes to support development in those languages. Perl includes 5739 lines of LISP, and Python includes another 2353 of LISP that is directly used to support elaborate Emacs modes for program editing. Other programs (such as the GIMP and Sawmill) also use LISP or one of its variants as a ``control'' language to control components built in other languages (in these cases C). LISP has a long history of use in the hacking (computer enthusiast) community, due to powerful influences such as MIT's old ITS community. For more information on the history of hackerdom, including the influence of ITS and LISP, see [Raymond 1999].

Some may be surprised at the number of different languages, but I believe this should be considered not a weakness but a strength. This GNU/Linux distribution supports a wide number of languages, enabling developers to choose the ``best tool for the job.''

3.4 Total Counts of Files

Of course, instead of counting SLOC, you could count just the number of files in various categories, looking for other insights.

Lex/flex and yacc/bison are widely-used program generators. They make respectable showings when counting SLOC, but their widespread use is more obvious when examining the file counts. There are 94 different lex/flex files, and 138 yacc/bison files. Some build directories use lex/flex or yacc/bison more than once.

Other insights can be gained from the file counts. There were 352,549 files, of which 130,488 were counted source code files (ignoring duplicate files and automatically generated ones). Not included in this count were 10,807 files which contained duplicate contents, and 1,587 files which were detected as being automatically generated.

These values can be used to compute average SLOC per file across the entire system. For example, for C, there was 21,461,450 SLOC contained in 78,676 files, resulting in an ``average'' C file containing 273 (14218806/52088) physical source lines of code. Intriguingly enough, Red Hat Linux 6.2 had essentially the same average number of physical lines of C code.

3.5 Total Counts by License

Here are the various license types, sorted by the SLOC in the packages with those licenses (see section 2.4 for how these license values were determined):

```
 15185987 (50.36%) GPL
  2498084 (8.28%)  MIT
  2305001 (7.64%)  LGPL
  2065224 (6.85%)  MPL
  1826601 (6.06%)  Distributable
  1315348 (4.36%)  BSD
   907867 (3.01%)  BSD-like
   766859 (2.54%)  Freely distributable
   692561 (2.30%)  Free
   455980 (1.51%)  GPL, LGPL
   323730 (1.07%)  GPL/MIT
   321123 (1.07%)  Artistic or GPL
   191379 (0.63%)  PHP
   173161 (0.57%)  Apache-like
   161451 (0.54%)  OpenLDAP
   146647 (0.49%)  LGPL/GPL
   103439 (0.34%)  GPL (programs), relaxed LGPL (libraries),
                   and public domain (docs)
   103291 (0.34%)  Apache
    73650 (0.24%)  W3C
    73356 (0.24%)  IBM Public License
    66554 (0.22%)  University of Washington's Free-Fork License
    59354 (0.20%)  Public domain
    39828 (0.13%)  GPL and Artistic
    31019 (0.10%)  GPL or BSD
    25944 (0.09%)  GPL/BSD
    20740 (0.07%)  Not listed
    20722 (0.07%)  MIT-like
    18353 (0.06%)  GPL/LGPL
    12987 (0.04%)  Distributable - most of it GPL
    8031 (0.03%)  Python
    6234 (0.02%)  GPL/distributable
    4894 (0.02%)  Freely redistributable
    1977 (0.01%)  Artistic
    1941 (0.01%)  GPL (not Firmware)
     606 (0.00%)  Proprietary
```

These can be grouped by totalling up SLOC for licenses containing certain key phrases:

```
16673212 (55.30%) GPL
 3029420 (10.05%) LGPL
 2842536 (9.43%)  MIT
 2612681 (8.67%)  distributable
 2280178 (7.56%)  BSD
 2065224 (6.85%)  MPL
  162793 (0.54%)  public domain
```

From these numbers, you can determine that:

1. The GPL is far and away the most common license (by lines of code) of any single license. In fact, the category ``GPL'' (packages with only this one license) all by itself accounts for 50.36% of the packages. By totalling the SLOC for all packages that include "GPL" in the license text, the total rises to 55%. No matter how you look at it, the GPL is the dominant single license in this distribution.
2. The next most common licenses were the LGPL, MIT, BSD, and MPL licenses (in order). This is in line with expectations: the most well-known and well-used open source licenses are the GPL, LGPL, MIT, and BSD licenses. Although the MPL does well in terms of SLOC, there is only one program in this distribution that uses it - Mozilla. There is some use of the ``Artistic'' license, but its use is far less; note that papers such as Perens [1999] specifically recommend against using the the Artistic license due to its legal ambiguities.
3. Very little software is released as public domain software (``no copyright''). In this distribution, only 0.2% of the software is in packages labelled as ``public domain'' (note that the 0.54% figure above includes the ``sane'' package which has documentation in the public domain). There may be several factors that account for this. First, if a developer wishes to get credit for their work, this is a poor ``license;'' by law anyone can claim ownership of ``public domain'' software. Second, there may be a fear of litigation; both the MIT and BSD licenses permit essentially arbitrary use but forbid lawsuits. While licenses such as MIT's and BSD's are not proof against a lawsuit, they at least provide some legal protection, while releasing software to the public domain provides absolutely no protection. Finally, any software released into the public domain can be modified and re-licensed under any other license, so there's nothing that keeps updated public domain software in the public domain.
4. There is a tiny amount of proprietary code, which is entirely in one component - Netscape Communicator / Navigator. This component uses the Motif toolkit (which is not open source) and has proprietary code mixed into it. As a result, almost none of the code for this package is is included on the CD-ROM - only a small amount of

``placeholder" code is there. In the future it is expected that this component will be replaced by Mozilla.

5. The packages which are clearly MIT-like/BSD-like licenses (totalling the MIT, BSD, MIT-like, BSD-like, and none/public domain entries) total 4,742,021 SLOC (15.92%). It's worth noting that 1,837,608 of these lines (39%) is accounted for by the XFree86 X server, an infrastructure component used for GNU/Linux's graphical user interface (GUI).
6. If the license types "distributable", "freely distributable", "MPL", "Free", "Artistic", "Apache", "Apache-like", and "IBM Public license" software was also considered MIT-like/BSD-like, the total SLOC would be 7,954,474 (26%, down from 36%). Unfortunately, the information to determine which of these other packages are simply BSD-like/MIT-like licenses is not included in the specification files.
7. The packages which include copylefting licenses (GPL or LGPL) total 63%. Limiting to only those that are GPL, LGPL, or both yields 60%, the same percentage as in Red Hat Linux 6.2 and a clear majority.

It is quite clear that in this distribution the GPL is the dominant license and that copylefting licenses (the GPL and LGPL) significantly outnumber the BSD/MIT-style licenses. This is a simple quantitative explanation why several visible projects (Mozilla, Troll Tech's Qt, and Python) have changed their licenses so that they're compatible with the GPL. When there is so much GPL software, GPL compatibility is critically important to the survival of many open source projects. See the Free Software Foundation's information on Various Licenses and Comments about Them [FSF 2001a] for information on GPL compatibility, and the GPL FAQ [FSF 2001b] for more information on the GPL in general.

The most common open source licenses in this distribution (by SLOC) are the GPL, MIT, LGPL, and BSD licenses (as well as the MPL, but note that it's only used by one project). Note that this is consistent with Perens [1999], who pleads that developers use an existing license instead of developing a new license where possible.

As of this writing, the GPL has received the most attention of these licenses, because Microsoft has specifically been attacking the GPL license. The GPL license permits commercial use of the program, but requires that distributors of modified versions of the program must also release the source code to their changes under the same terms. Therefore, software released under the GPL resists Microsoft's usual ``embrace and extend" approach to destroy competitors - Microsoft can use and change GPL'ed code, but it cannot make its changes to that code proprietary. As a counter-example, Kerberos (a security component released using an MIT license instead of the GPL) was recently incorporated by Microsoft into their products, and then extended in an incompatible way to prevent users from fully interoperating between products [Schneier 2000]. Had Kerberos been released under a GPL or LGPL license, this would have been much more difficult. The presence of so many GPL and LGPL components should make GNU/Linux distributions more resistant to being ``embraced, extended, and extinguished."

3.6 Total SLOC Counts

Given all of these assumptions, the counting programs compute a total of 30,152,114 physical source lines of code (SLOC); I will simplify this to ``over 30 million physical SLOC". This is an astounding amount of code; compare this to reported sizes of other systems:

Product	SLOC
NASA Space Shuttle flight control	420K (shuttle) + 1.4 million (ground)
Sun Solaris (1998-2000)	7-8 million
Microsoft Windows 3.1 (1992)	3 million
Microsoft Windows 95	15 million
Microsoft Windows 98	18 million
Microsoft Windows NT (1992)	4 million
Microsoft Windows NT 5.0 (as of 1998)	20 million
Red Hat Linux 6.2 (2000)	17 million

These numbers come from Bruce Schneier's *Crypto-Gram* [Schneier 2000], except for the Space Shuttle numbers which come from a National Academy of Sciences study [NAS 1996] and the Red Hat Linux 6.2 numbers which come from [Wheeler 2001]. Numbers for later versions of Microsoft products are not shown here because their values have great uncertainty in the published literature. The assumptions of most of these numbers are unclear (e.g., are these physical or logical lines of code?), but they are likely to be comparable physical SLOC counts.

Note that a deployed ``minimal system" would have less code; see the paper analyzing Red Hat Linux 6.2 for more discussion about this [Wheeler 2001].

Note that the Red Hat Linux 7.1 system includes a number of applications - in many cases a choice for each category. There are two major desktop environments (GNOME and KDE), plus various lightweight options. There are two word processors (Abiword and KWord), two spreadsheets (Gnumeric and KSpread), two relational database systems (MySQL and Postgres), and two web servers (Apache and TUX). In short, Red Hat Linux 7.1 includes a large number of applications, many of which are not included in its Microsoft or Sun equivalents.

At first blush, this bundling of applications with the operating system might appear similar to Microsoft's policy of combining applications with operating systems (which got Microsoft into legal trouble). However, it's worth noting some differences. First, and most important legally, a judge has ruled that Microsoft is a monopoly, and under U.S. law monopolies aren't allowed to perform certain actions that other organizations may perform. Second, anyone can take GNU/Linux, bundle it with an application, and redistribute the resulting product. There is no barrier such as ``secret interfaces" or relicensing costs that prevent anyone from making an application work on or integrate with GNU/Linux. Third, this distribution (and many others) include alternatives; users can choose between a number of options, all on the CD-ROM. Thus, while GNU/Linux distributions also appear to be going in the direction of adding applications to their system, they do not do so in a way that significantly interferes with a user's ability to select between alternatives.

It's worth noting that SLOC counts do not necessarily measure user functionality very well. For example, smart developers often find creative ways to simplify problems, so programs with smaller SLOC counts can sometimes provide greater functionality than programs with larger SLOC counts. However, there is evidence that SLOC counts correlate to effort (and thus development time), so using SLOC to estimate effort is still valid.

Creating reliable code can require much more effort than creating unreliable code. For example, it's known that the Space Shuttle code underwent rigorous testing and analysis, far more than typical commercial software undergoes, driving up its development costs. However, it cannot be reasonably argued that reliability differences between GNU/Linux and either Solaris or Windows NT would necessary cause GNU/Linux to take less effort to develop for a similar size. To see this, let's pretend that GNU/Linux had been developed using traditional proprietary means and a similar process to these other products. As noted earlier, experiments suggest that GNU/Linux, or at least certain portions of it, is more reliable than either. This would either cost more money (due to increased testing) or require a substantive change in development process (e.g., through increased peer review). Therefore, GNU/Linux's reliability suggests that developing GNU/Linux traditionally (at the same level of reliability) would have taken at least the same amount of effort if similar development processes were used as compared to similarly-sized proprietary systems.

3.7 Effort and Cost Estimates

Finally, given all the assumptions shown previously, the effort values are:

```
Total Physical Source Lines of Code (SLOC)                = 30152114
Estimated Development Effort in Person-Years (Person-Months) = 7955.75 (95469)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Estimated Schedule in Years (Months)                      = 6.53 (78.31)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Total Estimated Cost to Develop                           = $ 1074713481
 (average salary = $56286/year, overhead = 2.4).
```

See appendix A for more data on how these effort values were calculated; you can retrieve more information from http://www.dwheeler.com/sloc.

4. Conclusions

Red Hat Linux 7.1 includes over 30 million physical source lines of code (SLOC), compared to well over 17 million SLOC in version 6.2 (which had been released about one year earlier). Using the COCOMO cost model, this system is estimated to have required about 8,000 person-years of development time (as compared to 4,500 person-years to develop version 6.2).

Had this GNU/Linux distribution been developed by conventional proprietary means, it would have cost over $1.08 billion (1,000 million) to develop in the U.S. (in year 2000 dollars). Compare this to the $600 million estimate for version 6.2. Thus, Red Hat Linux 7.1 represents over a 60% increase in size, effort, and traditional development costs over Red Hat Linux 6.2. This is quite extraordinary, since this represents approximately one year.

This does not mean that all of the code added to the distribution in this thirteen month time period was actually written in that time period. In many cases, it represents the addition of whole new packages that have been in development for years, but have only now become sufficiently mature to include in the distribution. Also, many projects are developed over time and then released once testing is complete, and the time periods between releases can be more than a year. Still, from the user's point of view, this is a valid viewpoint - within one year of time much more functionality became available within their distribution.

Many other interesting statistics emerge. The largest components (in order) were the Linux kernel (including device drivers), Mozilla (Netscape's open source web system including a web browser, email client, and HTML editor), the X Window system (the infrastructure for the graphical user interface), gcc (a compilation system), gdb (for debugging), basic binary tools, emacs (a text editor and far more), LAPACK (a large Fortran library for numerical linear algebra), the Gimp (a bitmapped graphics editor), and MySQL (a relational database system). Note that Mozilla and LAPACK were not included in Red Hat 6.2 at all. Note that some projects (in particular KDE and GNOME) are in aggregate large enough to be one of the largest components, but because they are developed and distributed as a large number of smaller components, their totals don't appear in the list of largest components.

The languages used, sorted by the most lines of code, were C (71% - was 81%), C++ (15% - was 8%), shell (including ksh), Lisp, assembly, Perl, Fortran, Python, tcl, Java, yacc/bison, expect, lex/flex, awk, Objective-C, Ada, C shell, Pascal, and sed. C is still the predominant language but less so, with C++ primarily taking the difference. This appears in part to reflect many programmers' choice of C++ over C for GUI development. The increased amount of Fortran is primarily due to the inclusion of LAPACK.

The predominant software license is the GNU GPL. Slightly over half of the software is simply licensed using the GPL, and the software packages using the copylefting licenses (the GPL and LGPL), at least in part, accounted for 63% of the code. In all ways, the copylefting licenses (GPL and LGPL) are the dominant licenses in this GNU/Linux distribution. In contrast, only 0.2% of the software is in the public domain.

Clearly, this demonstrates that it is possible to build large-scale systems using open source approaches. Back in 1976, Bill Gates published his ``Open Letter to Hobbyists'', claiming that if software was freely shared it would prevent the writing of good software. He asked rhetorically, ``Who can afford to do professional work for nothing? What hobbyist can put three man-years into programming, finding all bugs, documenting his product, and distribute it for free?'' He presumed these were unanswerable questions, and both he and others based an industry on this assumption [Moody 2001]. Now, however, there are thousands of developers who are writing their own excellent code, and then giving it away. Gates was fundamentally wrong: sharing source code, and allowing others to extend it, is indeed a practical approach to developing large-scale systems - and its products can be more reliable.

It would be interesting to re-run these values on other GNU/Linux distributions (such as SuSE and Debian), and other open source systems (such as FreeBSD). SuSE and Debian, for example, by policy include many more packages, and would probably produce significantly larger estimates of effort and development cost.

As was noted in the previous paper, some actions by developers could simplify further similar analyses. The most important would be for programmers to always mark, at the top, any generated files (e.g., with a phrase like ``Automatically generated''). This would do much more than aid counting tools - programmers are likely to accidentally manually edit such files unless the files are *clearly* marked as files that should not be edited. It would be useful if developers would use file extensions consistently and not ``reuse'' extension names for other meanings; the suffixes(7) manual page lists a number of already-claimed extensions. This is more difficult for less-used languages; many developers have no idea that ``.m'' is a standard extension for objective-C. It would also be nice to have high-quality open source tools for performing logical SLOC counting on all of the languages represented here.

It should be re-emphasized that these are *estimates*; it is very difficult to precisely categorize all files, and some files might confuse the size estimators. Some assumptions had to be made (such as not including makefiles) which, if made differently, would produce different results. Identifying automatically-generated files is very difficult, and it's quite possible that some were miscategorized.

Nevertheless, there are many insights to be gained from the analysis of entire open source systems, and hopefully this paper has provided some of those insights. It is my hope that, since open source systems make it possible for anyone to analyze them, others will pursue many other lines of analysis to gain further insight into these systems.

More information is available at http://www.dwheeler.com/sloc.

Appendix A. Details of Approach

This appendix discusses some of the issues I had to deal with when performing the analysis, hopefully in enough detail that someone could repeat the effort.

In particular, installing the source code required two steps:

1. install the source code files (converting source RPM packages into "spec" files and compressed source files),
2. unpack the source code files (which generates uncompressed source code, and the licensing information),

I then ran sloccount version 1.9 to analyze the source code files, and examined the warning messages and fixed any serious problems that arose. This was not as easy as it sounds; the previous paper (analyzing Red Hat Linux 6.2) discusses this in more detail [Wheeler 2001]. I've since released the tools I used to count code as the program sloccount, available at http://www.dwheeler.com/sloccount.

One complication should be mentioned here. Although the Red Hat Linux 7.1 package comes with a CD-ROM labelled "Source Code", the "Source Code" CD-ROM doesn't contain all the source code. Skipping the source code on the "binary" CD-ROM would produce an invalid count, because 224 source code packages are placed there (including important packages like ssl, perl, python, and samba). It's likely this was done because of CD-ROM space needs - there is so much source code that, even when compressed, it doesn't fit on one CD-ROM.

I then searched for "old" versions of programs that were also included on the CD (so that the same program wouldn't be counted twice), or those required for non-Intel x86 operation (since these would not be fully counted anyway). I did this by examining any specification (in /usr/src/redhat/SPECS) with "compat" or "10" or "11" in its title (it turned out all of them were old and needed removing). I also examined anything ending in a digit or "x" followed by ".spec", which located qt1x.spec. Through this process I removed:

```
 compat-egcs.spec  compat-glibc.spec  compat-libs.spec  kde1-compat.spec
 gtk+10.spec libxml10.spec x86-compat-libs.spec qt1x.spec
```

I also removed any ``beta" software which had a non-beta version available (beta software was identified by searching for ``beta" in the package or specification file name). This removed:

```
  glib-gtkbeta.spec  gtk+-gtkbeta.spec pango-gtkbeta.spec
```

I also removed "mysqlclient9.spec". This specification contained the older MySQL client library version 3.23.22, as shipped with Red Hat Linux 7, for use with applications linked against it. I did include "mysql.spec", which had the code for the newer version 3.23.36 of MySQL (a relational database package).

Note that unlike Red Hat Linux 6.2, Red Hat Linux 7.1 didn't have two versions of bash or ncurses, so I didn't have to remove old versions of them. I left db1, db2, and db3 in, because it can be argued that none of these three necessarily replaces the other two.

One complication was in handling the graphical subsystem "XFree86". Version 4 of XFree86 was used for all client-side applications, but Red Hat uses both version 3 and version 4 to implement various X servers. I looked at the XFree86 source package for version 4, and it turned out that server code was included in the package. Rather than have XFree86 counted essentially twice (once as version 3, and another as version 4), I only counted the code in version 4 of XFree86. This could be argued both ways; I understand that version 4 is a massive rewrite of much of the version 3 server, so counting it twice is actually not irrational. And unintentionally, I ended up counting a small amount of version 3 code through reuse by another program. It turns out that vnc_unixsrc includes (through reuse) portions of the X Window system version 3 code; in their words, ``a cut-down version of the standard XFree86 distribution (``server only" distribution) without many of the later X extensions or hardware-specific code." VNC won't work without that code, and clearly there was effort to build version 3 and to rebuild version 4, so I let these counts stand.

I then unpacked the source code by running code that in essence did this:

```
  cd /usr/src/redhat/SPECS
  rpm -bp *.spec
```

This uncompresses the source code and applies all patches used by the actual system. Since I wanted to count the amount of code actually included in the system, it was important to include the patches. The actual code to unpack the source code was more complex, because it also marked every unpacked directory (in the BUILD directory) to identify the spec file it came from and the license of the program. The license was determined by (1) looking at the "Copyright" and "License" fields of the spec file, and if that didn't work, (2) looking at various files in the build directory, such as "LICENSE", "COPYING*", and "Artistic". Unfortunately, MIT-like and BSD-like licenses can be harder to detect (because their text can be varied), but many licenses (such as the GPL and LGPL) can be detected with great confidence. I used the "spec" file as the primary source, because this was placed by a human (who could better understand legal technicalities than a machine).

I actually had to repeat the unpacking more than once; the RPM system would notice a missing dependency for building the software and protest. This required installation of the missing component (in some cases I didn't have to install the program and could have forced installation, but I did not want to risk corrupting the results by failing to install a package).

A surprising development was that the packages "imap" and "samba" reported errors in unpacking. For imap, patch #5 (imap-4.7c2-flock.patch) and for samba, patch #21 (samba-ia64.patch of source/passdb/pass_check.c) would cause unpacking to halt. I unpacked the software and simply counted what was there; this appears to be what the original developers did.

I examined the reported license values, in particular for all code more than 100,000 source lines of code (as the largest components, wrong values for these components would be more likely to cause significant error). I found that Perl had been assigned "GPL" in its spec file, but this isn't the whole story; as documented in its README file, Perl can be used under either the GPL or Artistic license, so its license entry was changed to "GPL or Artistic". Mozilla's licensing situation is more complex; some portions of it are actually under a separate dual licensing scheme (licensed under both the GPL and Netscape Public License, i.e., NPL). However, labelling it as "MPL, NPL, and GPL" would probably overstate the amount of code licensed under the GPL, so I left its entry as the MPL license.

Note that the unpacked source files (including source code, fonts, documentation, and so on) totalled more than 4.4 Gigabytes.

I ran the analysis code as a normal user, so I first had to set the permissions for users to read the code. I then reverted to normal user account, and used sloccount version 1.9 to measure the source code, using the following bash command:

```
sloccount --multiproject /usr/src/redhat/BUILD > sloc-actions 2>&1 &
```

Note that I did _not_ use the "--follow" option of sloccount. Some programs, notably pine, include a symbolic link to other directories such as /usr/lib. Thus, using --follow would have included files outside of the intended directory in the analysis.

I looked over various error reports and determined that none would fundamentally invalidate the results. For example, there were several errors in the XFree86 source code involving improperly formatted strings. It appears that these are syntax errors in the code that are preprocessed away (and thus not noticed by the compiler). I intend to report these problems to the XFree86 project. One program was a bash shell script that began with "#! /usr/bin/env bash", which sloccount's heuristics could not handle at the time. I then modified sloccount to correctly determine its type (it's a bash shell script).

Note that sloccount creates a large number of small files. This isn't fundamentally a problem, but because of the large scale of the system I found that I ran out of inodes if I tried to store multiple copies of results. Those who try to duplicate this activity may want to specially format their filesystems to include more inodes.

For a complete list of all components and their SLOC counts, see http://www.dwheeler.com/sloc/redhat71-v1/summary.

References

[Boehm 1981] Boehm, Barry. 1981. *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall, Inc. ISBN 0-13-822122-7.

[Dempsey 1999] Dempsey, Bert J., Debra Weiss, Paul Jones, and Jane Greenberg. October 6, 1999. UNC Open Source Research Team. Chapel Hill, NC: University of North Carolina at Chapel Hill. http://www.ibiblio.org/osrt/develpro.html.

[DSMC] Defense Systems Management College (DSMC). *Indirect Cost Management Guide: Navigating the Sea of Overhead*. Defense Systems Management College Press, Fort Belvoir, VA 22060-5426. Available as part of the ``Defense Acquisition Deskbook." http://web2.deskbook.osd.mil/reflib/DTNG/009CM/004/009CM004DOC.HTM

[FSF 2000] Free Software Foundation (FSF). *What is Free Software?*. http://www.gnu.org/philosophy/free-sw.html.

[FSF 2001a] Free Software Foundation (FSF). 2001. *Various Licenses and Comments about Them*. http://www.gnu.org/philosophy/license-list.html.

[FSF 2001b] Free Software Foundation (FSF). 2001. *General Public License (GPL) Frequently Asked Questions (FAQ)* http://www.gnu.org/copyleft/gpl-faq.html.

[Godfrey 2000] Godfrey, Michael W., and Qiang Tu. Software Architecture Group (SWAG), Department of Computer Science, University of Waterloo. ``Evolution in Open Source Software: A Case Study." *2000 Intl Conference on Software Maintenance.* http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf

[Halloween I] Valloppillil, Vinod, with interleaved commentary by Eric S. Raymond. Aug 11, 1998. "Open Source Software: A (New?) Development Methodology" v1.00. http://www.opensource.org/halloween/halloween1.html.

[Halloween II] Valloppillil, Vinod and Josh Cohen, with interleaved commentary by Eric S. Raymond. Aug 11, 1998. "Linux OS Competitive Analysis: The Next Java VM?". v1.00. http://www.opensource.org/halloween/halloween2.html

[Kalb 1990] Kalb, George E. "Counting Lines of Code, Confusions, Conclusions, and Recommendations". Briefing to the 3rd Annual REVIC User's Group Conference, January 10-12, 1990. http://sunset.usc.edu/research/CODECOUNT/documents/3rd_REVIC.pdf

[Kalb 1996] Kalb, George E. October 16, 1996 "Automated Collection of Software Sizing Data" Briefing to the International Society of Parametric Analysts, Southern California Chapter. http://sunset.usc.edu/research/CODECOUNT/documents/ispa.pdf

[Masse 1997] Masse, Roger E. July 8, 1997. *Software Metrics: An Analysis of the Evolution of COCOMO and Function Points*. University of Maryland. http://www.python.org/~rmasse/papers/software-metrics.

[Miller 1995] Miller, Barton P., David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, and Jeff Steidl. 1995. *Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services*. http://www.cs.wisc.edu/~bart/fuzz/fuzz.html.

[Moody 2001] Moody, Glyn. 2001. *Rebel Code*. ISBN 0713995203.

[NAS 1996] National Academy of Sciences (NAS). 1996. *Statistical Software Engineering*. http://www.nap.edu/html/statsoft/chap2.html

[Netcraft 2001] Netcraft. September 2001. Netcraft Web Server Survey. http://www.netcraft.com/Survey/index-200109.html.

[OSI 1999]. Open Source Initiative. 1999. *The Open Source Definition*. http://www.opensource.org/osd.html.

[Park 1992] Park, R. 1992. *Software Size Measurement: A Framework for Counting Source Statements*. Technical Report CMU/SEI-92-TR-020. http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.020.html

[Perens 1999] Perens, Bruce. January 1999. *Open Sources: Voices from the Open Source Revolution*. "The Open Source Definition". ISBN 1-56592-582-3. http://www.oreilly.com/catalog/opensources/book/perens.html

[Raymond 1999] Raymond, Eric S. January 1999. ``A Brief History of Hackerdom''. *Open Sources: Voices from the Open Source Revolution*. http://www.oreilly.com/catalog/opensources/book/raymond.html.

[Raymond 2001] Raymond, Eric S. February 22, 2001. *Software Release Practice HOWTO*. http://www.linuxdoc.org/HOWTO/Software-Release-Practice-HOWTO/index.html

[Schneier 2000] Schneier, Bruce. March 15, 2000. ``Software Complexity and Security''. *Crypto-Gram*. http://www.counterpane.com/crypto-gram-0003.html

[Shankland 2000a] Shankland, Stephen. February 14, 2000. "Linux poses increasing threat to Windows 2000". CNET News.com. http://news.cnet.com/news/0-1003-200-1549312.html.

[Shankland 2000b] Shankland, Stephen. August 31, 2000. "Red Hat holds huge Linux lead, rivals growing". CNET News.com. http://news.cnet.com/news/0-1003-200-2662090.html

[Stallman 2000] Stallman, Richard. October 13, 2000 "By any other name...". http://www.anchordesk.co.uk/anchordesk/commentary/columns/0,2415,7106622,00.html.

[Vaughan-Nichols 1999] Vaughan-Nichols, Steven J. Nov. 1, 1999. Can you Trust this Penguin? ZDnet. http://www.zdnet.com/sp/stories/issue/0,4537,2387282,00.html

[Wheeler 2000a] Wheeler, David A. 2000. *Open Source Software / Free Software References*. http://www.dwheeler.com/oss_fs_refs.html.

[Wheeler 2000b] Wheeler, David A. 2000. *Quantitative Measures for Why You Should Consider Open Source / Free Software*. http://www.dwheeler.com/oss_fs_why.html.

[Wheeler 2001]. Wheeler, David A. May 9, 2001 (minor update from November 6, 2000). *Estimating Linux's Size*. Version 1.04. http://www.dwheeler.com/sloc.

[Zoebelein 1999] Zoebelein. April 1999. http://leb.net/hzo/ioscount.

Trademark owners own any trademarks mentioned.

This paper is (C) Copyright 2001 David A. Wheeler. All rights reserved. You may download and print it for your own personal use, and of course you may link to it. Talk to me to request republication rights. When referring to the paper, please refer to it as ``More than a Gigabuck: Estimating GNU/Linux's Size" by David A. Wheeler, located at http://www.dwheeler.com/sloc. Please give credit if you refer to any of its techniques or results.