(S)LOC Count Evolution for Selected OSS Projects

Tik Report 315

Arno Wagner
arno@wagner.name

December 11, 2009

Abstract

We measure the dynamics in project code size for several large open source projects, given as lines of code. Lines of codes are counted using two different methods. First, raw LOCs, which count every line in the decompressed source distribution package and second, SLOCs, as defined by David A. Wheeler's 'SLOCCount'. It turns out that the results are substantially different but show similar trends in many cases. A primary result is that most projects have had steadily increasing code size, with the associated increased project complexity. We also briefly discuss the relevance of LOCs in comparison to other code complexity measures.

1 Introduction

Software complexity metrics are an important factor in many investigations. They are used for cost and time estimation in project planning, for the estimation on how many coding errors (bugs) and vulnerabilities are to be expected in a project.

There are numerous metrics in use and even a simple measure such as "lines of code" (LOCs) is used in vastly different flavours. Our goal is to measure the LOC-based complexity of several open source projects over a longer period of time with exactly defined and documented metrics. For this we use two popular counting methods, namely raw LOCs and SLOCs. While these measures have limited applicability when comparing two different projects, they seem well suited to document the evolution of a single project over time. This report is structured as follows: In Section 2 we define and compare the two different LOC measurements we are using and compare LOCs to other complexity measures. Section 3 presents the measurement results for four large open source projects, spanning a time period of up to 10 years. The remainder of the report surveys related work and gives concluding remarks. The exact numerical LOC counts found are documented in the appendix.

2 Lines of Code: Definition and Properties

2.1 Raw LOCs vs SLOCs

Raw LOCs are derived in the simplest possible manner, namely by taking the whole source distribution package of a project and counting every line ending in it. This is typically done by using the Unix command ”wc” on the decompressed tar file. While this measure seems overly simplistic, it is used in practice.

SLOCs [13] were defined by David A. Wheeler in order to get a more realistic line measurement. They are measured with the tool `sloccount`. Primary improvements are duplicate detection, separation of the results into different programming languages and identification of documentation in file-embedded as well as separate file format. Note that some people use SLOC as a synonym for LOC. In this report, SLOC always means SLOC as defined by Wheeler.

Naturally, raw LOC counts will be higher than SLOC counts. There is a tendency that whenever the sheer size of a project is stressed, raw LOCs are used, e.g. in Table 4 of [10]. We believe that this is the wrong approach. Especially for documentation-heavy projects or when documentation is suddenly included in the source distribution package to a far larger degree than before, raw LOC counts distort the actual project evolution. Our measurements show that in some cases project complexity measures based on raw LOCs can be quite different from those based on SLOCs. The problem is made worse by our observation that many publicly stated LOC counts are missing a description on how they were obtained. Still, our measurement shows that for 3 out of 4 cases we examined, raw LOCs actually closely follow the same trends as SLOCs over the project duration. The one deviation (Apache) is likely due to a change in the documentation style and inclusion in the source distribution package.

One factor SLOC cannot account for is code compactness. Due to different coding styles code with the same functionality and inherent complexity can be distributed over a larger or smaller number of lines. If a project has a style guide that is followed by the developers this variation can be reduced between different developers. This makes LOC counts a good tool to estimate project evolution over time. Comparing LOC counts of different projects is still problematic.

A second source of different LOC counts for similar functionality can be dues to use of different programming languages. For example things that can be expressed in Perl in very few lines may take hundreds of lines of code in C. Comparisons between different languages based on LOC counts are therefore highly problematic and should be avoided.

2.2 LOC Derived Complexity Measures

To infer actual complexity, LOC counts are often used as input into more complex models. One such model is the the Constructive Cost Model (COCOMO) [6] by Barry W. Boehm. The basic COCOMO estimates overall project effort E as

$$E[\text{Man-Months}] = a * (LOC/1000)^b$$

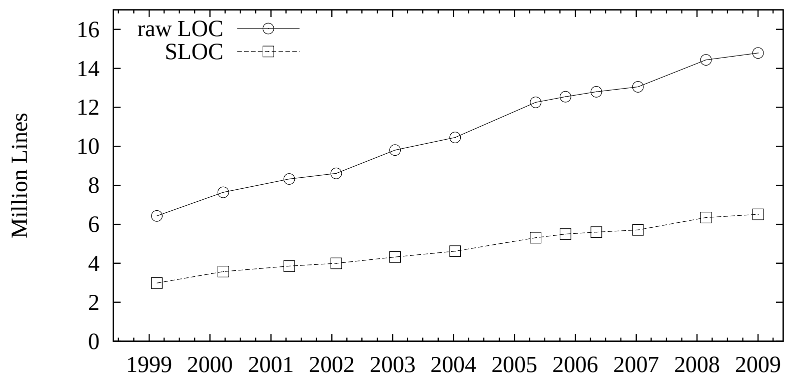

Figure 1: FreeBSD Code Size

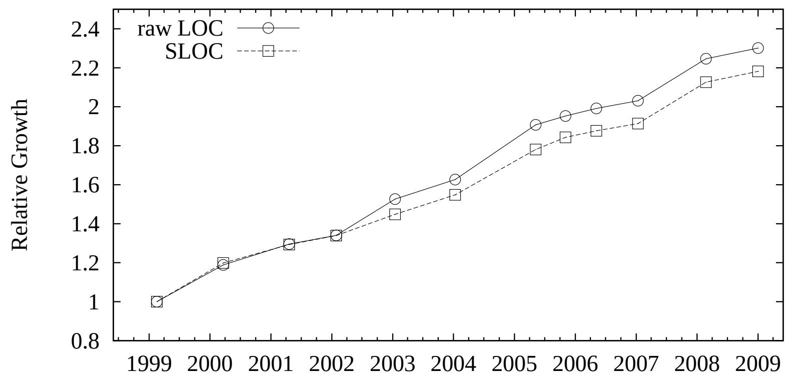

Figure 2: FreeBSD Normalized Code Growth

with $a \in \{2.4 \ldots 3.6\}$ and $b \in \{1.05 \ldots 1.2\}$ depending on team experience and size. The intermediate COCOMO also takes into account other project factors to modify factor a, while b is left unchanged. The COCOMO has limited accuracy [9] but is still suitable for rough estimates. Its claim that effort raises exponentially with the project size is not in doubt. An improved version is the COCOMO II [5, 2], which also takes aspects like code reuse and project state into account and offers better accuracy later in the design process.

2.3 Alternatives to LOCs

LOC counts suffer from being a simplistic measure. They do not take structural complexity into account. An alternative measure is *Cyclomatic Complexity* [11], which is based on the number of "decision points" in the code. Newer research found empirical evidence of a linear relationship between LOC counts and Cyclomatic Complexity [8]. This indicates that the LOC number of a software project is a significantly more meaningful than its simple nature suggests.

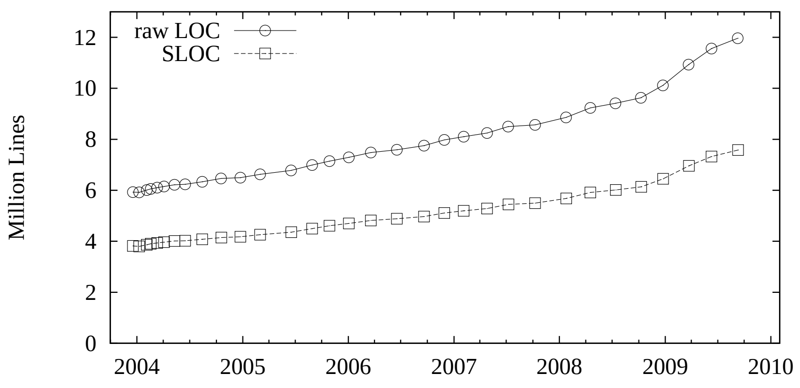

Figure 3: Linux Kernel Code Size

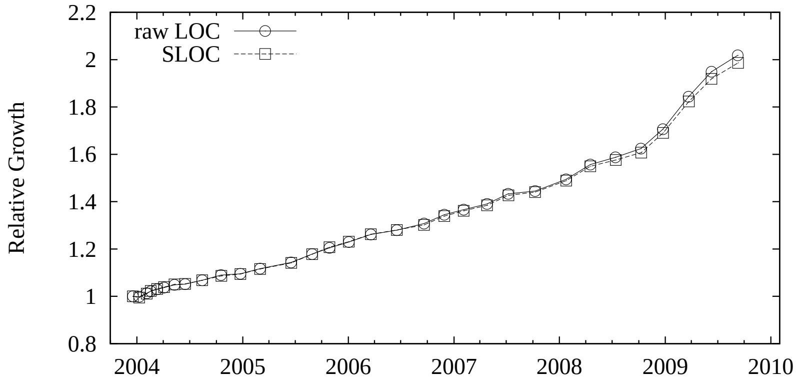

Figure 4: Linux Kernel Normalized Code Growth

3 Findings

For each project we give time plots for both raw LOCs and SLOCs in absolute and normalized form. The normalized forms are then compared in Figures 11 and 12. The exact numerical values are stated in Appendix A. The raw LOC numbers were obtained by running the Linux `wc` command on the decompressed tar archive. Sample tests showed that the results differ very little from those obtained when completely unpacking the tar archives, running `wc` on each individual file and then adding the individual counts. SLOCs were obtained by running `sloccount` on the unpacked distribution archive and taking the `Total Physical Source Lines of Code (SLOC)` output number. Note that running several instances of `sloccount` in parallel requires special attention, see section 3.5.

3.1 FreeBSD

The FreeBSD source packages were obtained from [1]. The packages come in a broken-down compressed tar format, that contains the complete source distribution but no binaries. Most of the source code is C. The LOC counts we

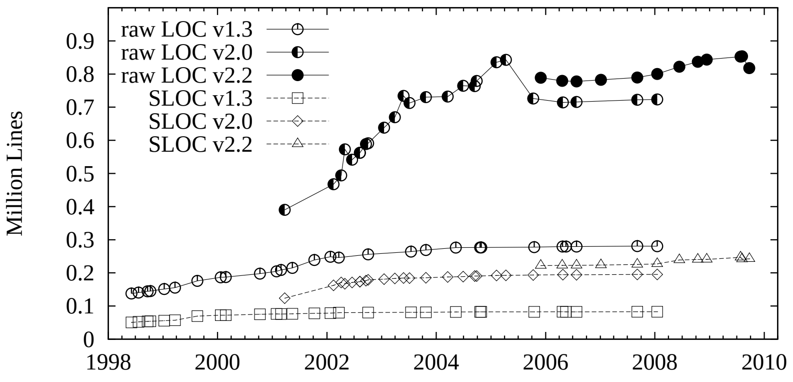

Figure 5: Apache Web Server Code Size

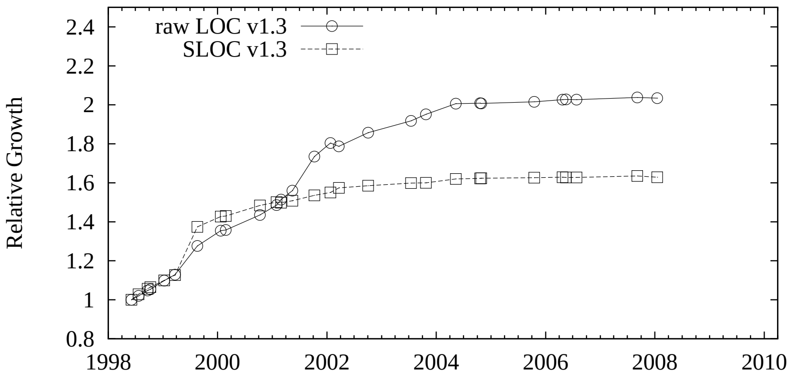

Figure 6: Apache 1.3 Web Server Normalized Code Growth

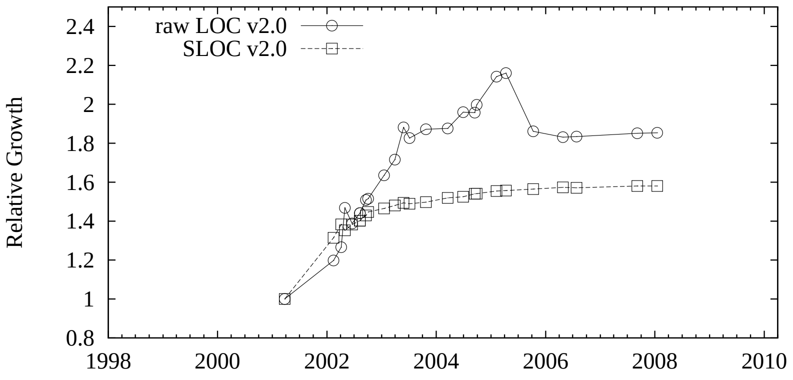

Figure 7: Apache 2.0 Web Server Normalized Code Growth

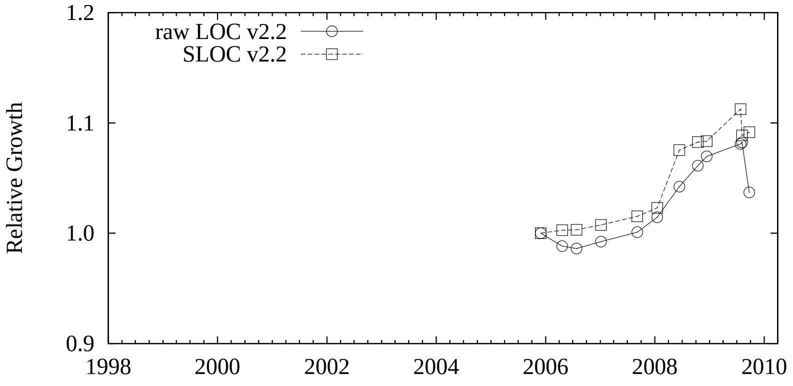

Figure 8: Apache 2.2 Web Server Normalized Code Growth

obtained are represented in Figures 1 and 2. They show a steady, near-linear, increase over time, with an almost constant ratio between raw LOC and SLOC numbers.

3.2 The Linux Kernel

The kernel packages used are stock kernel sources from `kernel.org`. We only measure 2.6.x versions, i.e. the respective initial kernel releases. Measurements on patched sub-releases are problematic, because they may actually have been released later than the next initial release kernel and they can contain a smaller or larger amount of back-ported code from later kernels. We therefore believe that measurements restricted to initial releases give the most realistic results.

The LOC value plots are given in Figure 3 and 4. Kernel growth accelerated in 2008. It is too early to see whether this was a temporary phenomenon or represents a persistent change. As for FreeBSD, raw LOC and SLOC changes are very similar for the Linux Kernel.

3.3 The Apache Web Server

The Apache sources are from the Apache archive download site at [1]. We did measurements on versions 1.3.x, 2.0.x and 2.2.x. Note that Apache 2.0.32 is a beta release and that 2.0.15 is an alpha release. The LOC plots for Apache 1.3, 2.0 and 2.2 are given in in graphical from in Figure 5 for the absolute values and in Figures 6, 7 and 8 in normalized form.

As the plots clearly show, there is a significant difference in the evolution of raw LOCs compared to SLOCs for the Apache web server. While the SLOC counts for all three versions increase slowly initially and then level off, the raw LOC counts show a much less steady behaviour with a sharp increase for version 2.0 until 2005 and a reduction afterwards, before getting constant. Version 2.2. has a more steady raw LOC development, but the raw LOC count remains far higher than the SLOC count.

The most likely explanation for the raw LOC and SLOC differences is that the amount of documentation shipped with the distribution package was significantly increased with the development of the 2.0.x Apache versions, and then kept up with the 2.2.x releases. The example of the Apache web server shows that raw LOC counts can be quite deceiving with regard to code growth.

3.4 The Firefox Web Browser

The Firefox source packages were obtained from the Mozilla download server at [3]. The LOC plots are given in Figures 9 and 10. Surprisingly the SLOC numbers stayed nearly constant for Firefox in the measurement interval. We can only speculate that this is due to new functionality being implemented in the form of plug-ins that are not distributed with the browser source package.

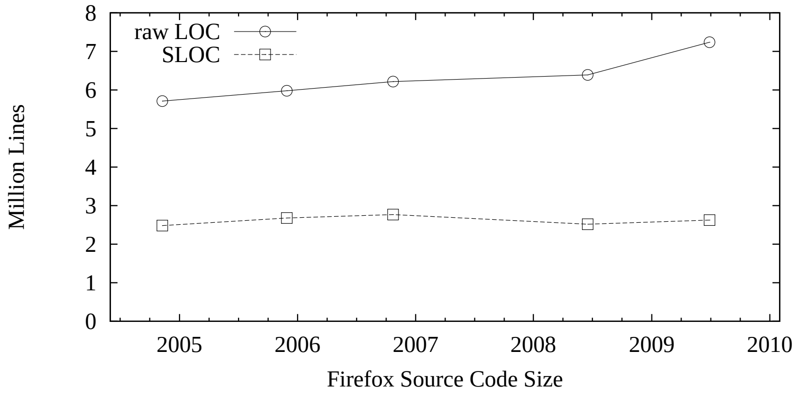

Figure 9: Firefox Web Browser Code Size

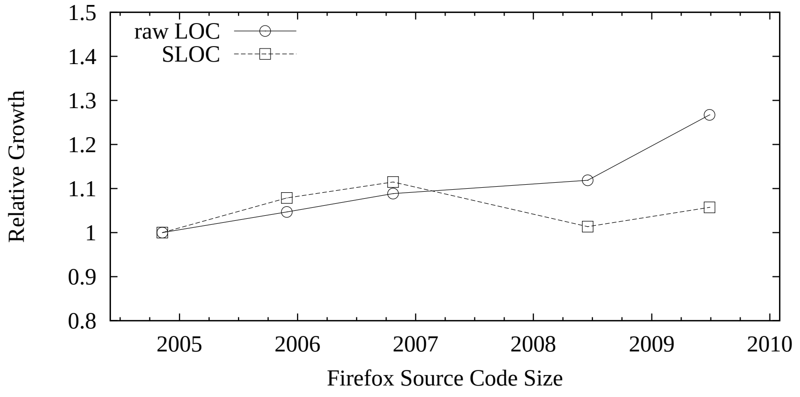

Figure 10: Firefox Web Browser Normalized Code Growth

3.5 Problems Found During Measurement

`sloccount` stores temporary data in `${HOME}/.slocdata`, but unfortunately fails to protect this directory against multiple use by two or more `sloccount` instances. Running two or more instances of `sloccount` in parallel results in random, hard to debug failures and sometimes in grossly wrong results. This either has to be avoided, or all instances running in parallel have to be given their own data directories using the `-datadir` option. It should be noted that `sloccount` leaves cleaning up the data directory to the user.

4 Conclusion

4.1 Related Work

Other code size count statistics have been published. For example [4] gives eLOC count for specific versions of several large Open Source projects. The authors of [10] use raw LOCs to give the size of the Linux kernel. David A. Wheeler gives SLOC counts for a complete Linux distribution (Red Hat Linux 7.1, vintage 2001) in [14]. This inspired a number of follow-up measurements

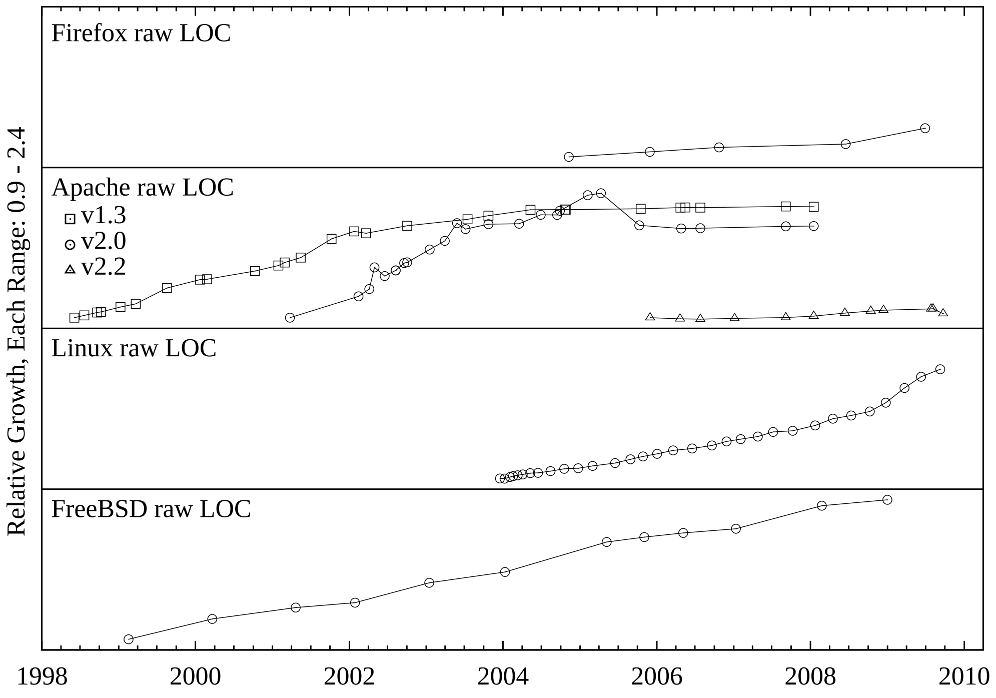

Figure 11: Normalized raw LOC Code Growth Comparison

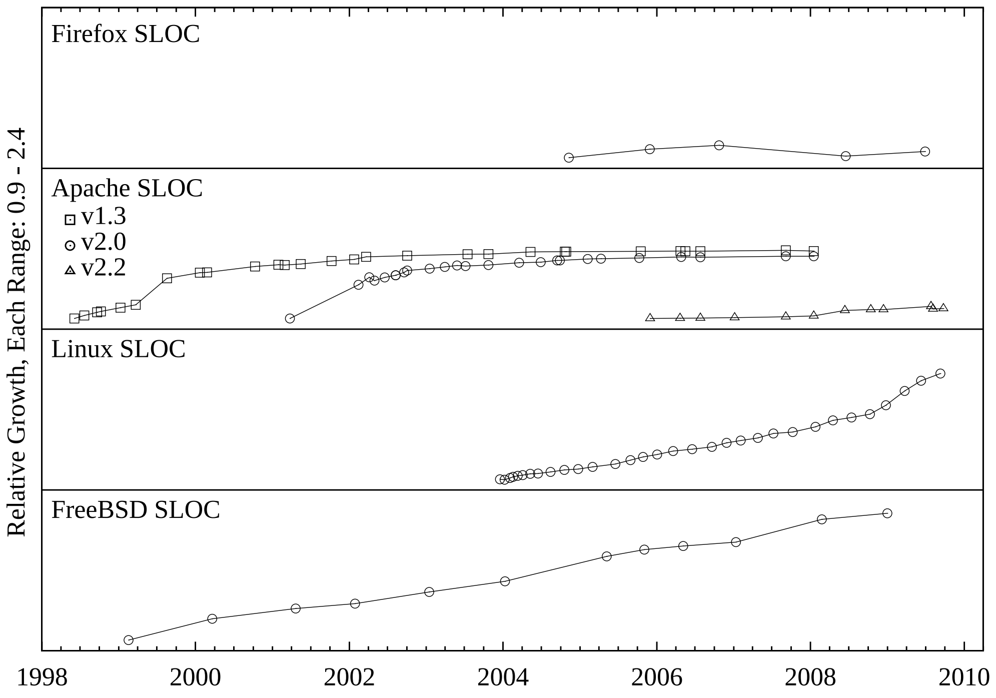

Figure 12: Normalized SLOC Code Growth Comparison

of specific versions of OSS projects or distributions, see [12].

The only long-term study of code size dynamics we are aware of is [7], were the aggregated growth of 5122 OSS projects on SourceForge for the time from 1995 to 2006 was measured in SLOC. It found an overall exponential growth.

4.2 Discussion

Our measurements show a steady increase in size for several large and widely used open source projects. The exception is the Firefox web browser, that has a slowly increasing raw LOC size, but an almost constant SLOC code size over a period of 4 years. A possible explanation is a shift of functionality to plug-ins that are not distributed directly with the core browser source code. For the other projects a significant increase in size, and hence code complexity, can be observed.

We also found that while raw LOC counts can sometimes precisely represent relative project code growth they are sensitive to inclusion of project documentation in the source distribution package and can give severely misleading numbers. For this reasons, SLOC or a similar method should be used to estimate code size and raw LOC counts should be avoided.

References

[1] Apache download archive. `http://archive.apache.org/dist/httpd/`.

[2] Cocomo ii. `http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html`. Last visited November, 2009.

[3] Mozilla download server. `ftp://ftp.mozilla.org/pub/mozilla.org/firefox/releases/`.

[4] RSM Metrics of Popular Software Programs. `http://msquaredtechnologies.com/m2rsm/rsm_software_project_metrics.htm`. Last visited November, 2009.

[5] B. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software cost estimation with COCOMO II*. Prentice-Hall, 2000.

[6] B. W. Boehm. *Software engineering economics*. Prentice-Hall, 1981.

[7] A. Deshpande and D. Riehle. The Total Growth of Open Source. In *Proceedings of the Fourth Conference on Open Source Systems (OSS 2008)*. Springer Verlag, 2008.

[8] G. Jay, J. E. Hale, R. K. Smith, D. Hale, N. A. Kraft, and C. Ward. Cyclomatic complexity and lines of code: empirical evidence of a stable linear relationship. *Journal of Software Engineering and Applications (JSEA)*, 2009.

[9] C. F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, May 1987.

[10] G. Kroah-Hartman, J. Corbet, and A. McPherson. Linux Kernel Development, August 2009 update. available from `http://www.linuxfoundation.org/publications`, 2009. Published by the Linux Foundation.

[11] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 1976.

[12] D. A. Wheeler. Counting Source Lines of Code (SLOC). `http://www.dwheeler.com/sloc/`. Last visited November, 2009.

[13] D. A. Wheeler. SLOCCount. `http://www.dwheeler.com/sloccount/`. Last visited November, 2009.

[14] D. A. Wheeler. More Than a Gigabuck: Estimating GNU/Linux's Size. `http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html`, 2002. Last visited November, 2009.

A Detailed LOC Counts

Version	Date	raw LOC	SLOC
3.1	1999-02-17	6425285	2983053
4.0	2000-03-21	7637813	3575773
4.3	2001-04-21	8323520	3858150
4.5	2002-01-28	8611733	3995585
5.0	2003-01-16	9808358	4320407
5.2	2004-01-11	10452114	4618569
5.4	2005-05-08	12255382	5312235
6.0	2005-11-03	12545630	5498400
6.1	2006-05-07	12796877	5598105
6.2	2007-01-12	13048819	5708606
7.0	2008-02-24	14431462	6343397
7.1	2009-01-01	14784426	6508506

Table 1: FreeBSD (S)LOC counts

Version	Date	raw LOC	SLOC
2.6.0	2003-12-18	5929913	3816370
2.6.1	2004-01-09	5919671	3797257
2.6.2	2004-02-04	6008957	3860620
2.6.3	2004-02-18	6056561	3903707
2.6.4	2004-03-11	6105182	3935151
2.6.5	2004-04-04	6149724	3964541
2.6.6	2004-05-10	6218461	4008692
2.6.7	2004-06-16	6236264	4017682
2.6.8	2004-08-14	6333645	4076622
2.6.9	2004-10-18	6463002	4147093
2.6.10	2004-12-24	6495542	4176875
2.6.11	2005-03-02	6624076	4257157
2.6.12	2005-06-17	6777861	4356161
2.6.13	2005-08-29	6988801	4496659
2.6.14	2005-10-28	7143234	4609589
2.6.15	2006-01-03	7290071	4697435
2.6.16	2006-03-20	7480063	4818320
2.6.17	2006-06-18	7588015	4886152
2.6.18	2006-09-20	7752847	4968235
2.6.19	2006-11-29	7976222	5111085
2.6.20	2007-02-04	8102534	5195239
2.6.21	2007-04-26	8246518	5284774
2.6.22	2007-07-08	8499411	5445218
2.6.23	2007-10-09	8566607	5497052
2.6.24	2008-01-24	8859684	5682749
2.6.25	2008-04-17	9232542	5913441
2.6.26	2008-07-13	9411791	6015867
2.6.27	2008-10-09	9630024	6133830
2.6.28	2008-12-24	10115663	6450761
2.6.29	2009-03-23	10930803	6958954
2.6.30	2009-06-10	11557330	7323310
2.6.31	2009-09-09	11966483	7581069

Table 2: Linux Kernel (S)LOC counts

Version	Date	raw LOC	SLOC
1.3.0	1998-06-05	137800	50712
1.3.1	1998-07-22	140609	52139
1.3.2	1998-09-21	144393	53598
1.3.3	1998-10-09	145214	54000
1.3.4	1999-01-10	151376	55778
1.3.6	1999-03-23	155565	57147
1.3.9	1999-08-19	175888	69683
1.3.11	2000-01-22	186657	72370
1.3.12	2000-02-25	187163	72517
1.3.14	2000-10-10	197723	75268
1.3.17	2001-01-29	204752	76128
1.3.19	2001-02-28	208633	75980
1.3.20	2001-05-15	214979	76466
1.3.22	2001-10-09	239025	77893
1.3.23	2002-01-24	248604	78625
1.3.24	2002-03-21	246255	79830
1.3.27	2002-10-03	255910	80392
1.3.28	2003-07-17	264314	81089
1.3.29	2003-10-24	268915	81137
1.3.31	2004-05-11	276415	82149
1.3.32	2004-10-21	276794	82314
1.3.33	2004-10-28	276632	82325
1.3.34	2005-10-17	277743	82481
1.3.35	2006-04-24	279274	82607
1.3.36	2006-05-17	279450	82534
1.3.37	2006-07-27	279309	82526
1.3.39	2007-09-06	280793	82923
1.3.41	2008-01-17	280329	82603

Table 3: Apache 1.3 (S)LOC counts

Version	Date	raw LOC	SLOC
2.0.15	2001-03-25	390164	123642
2.0.32	2002-02-14	467406	162540
2.0.35	2002-04-06	494125	171025
2.0.36	2002-05-01	572745	167244
2.0.39	2002-06-18	541670	170972
2.0.40	2002-08-09	562212	173366
2.0.40	2002-08-09	562212	173366
2.0.42	2002-09-19	588537	176743
2.0.43	2002-10-03	591149	178864
2.0.44	2003-01-18	638078	181144
2.0.45	2003-03-31	669498	183073
2.0.46	2003-05-28	733980	184721
2.0.47	2003-07-07	712578	184123
2.0.48	2003-10-24	730244	185183
2.0.49	2004-03-18	731918	187832
2.0.50	2004-06-29	764540	188573
2.0.51	2004-09-15	763639	190487
2.0.52	2004-09-28	779206	190535
2.0.53	2005-02-07	835689	192191
2.0.54	2005-04-11	842893	192540
2.0.55	2005-10-10	726140	193444
2.0.58	2006-04-27	714459	194553
2.0.59	2006-07-27	715738	194266
2.0.61	2007-09-06	722264	195392
2.0.63	2008-01-17	723272	195392

Table 4: Apache 2.0 (S)LOC counts

Version	Date	raw LOC	SLOC
2.2.0	2005-11-30	788634	222029
2.2.2	2006-04-22	779381	222631
2.2.3	2006-07-27	777676	222728
2.2.4	2007-01-06	782621	223688
2.2.6	2007-09-06	789395	225438
2.2.8	2008-01-17	800087	227133
2.2.9	2008-06-13	821917	238769
2.2.10	2008-10-14	836881	240374
2.2.11	2008-12-13	843579	240557
2.2.12	2009-07-27	852451	246981
2.2.13	2009-08-06	853515	241716
2.2.14	2009-09-23	817829	242365

Table 5: Apache 2.2 (S)LOC counts

Version	Date	raw LOC	SLOC
1.0	2004-11-09	5712472	2482532
1.5	2005-11-29	5980193	2677615
2.0	2006-10-24	6218536	2767806
3.0	2008-06-17	6390796	2516373
3.5	2009-06-29	7239320	2624988

Table 6: Firefox (S)LOC counts