Random statistics about BitlBee

Sjoerd Hemminga

December 4th, 2005

Abstract

This article shows some random statistics about BitlBee at the time of its 1.0 release. No conclusions or interpretations are given. Just tables and graphs.

1 Development days per release

Releases generally take a lot of time to develop. To quantify this amount of time we'll coin the term "development days," which we'll define as the number of days between two releases. For the first release, the number of development days is the number of days between the idea and the release.

Table 1 contains the development days for all BitlBee releases up to 1.0. I've based the release dates on the `doc/CHANGES` file in the BitlBee 1.0 release. The release date of 0.70 is based on the date the project was added to Freshmeat[1].

The data is plotted in figure 1.

Version	Days	Cumulative
0.70	47	47
0.71	37	84
0.72	94	178
0.73	115	293
0.74	58	351
0.74a	1	352
0.80	13	365
0.81	113	478
0.81a	1	479
0.82	15	494
0.83	61	555
0.84	44	599
0.85	29	628
0.85a	11	639
0.90	58	697
0.90a	38	735
0.91	89	824
0.92	151	975
0.93	250	1225
0.93a	0	1225
0.99	3	1228
1.0	31	1259
Average	57	–

Table 1: The number of development days per release.

[1] http://www.freshmeat.net/

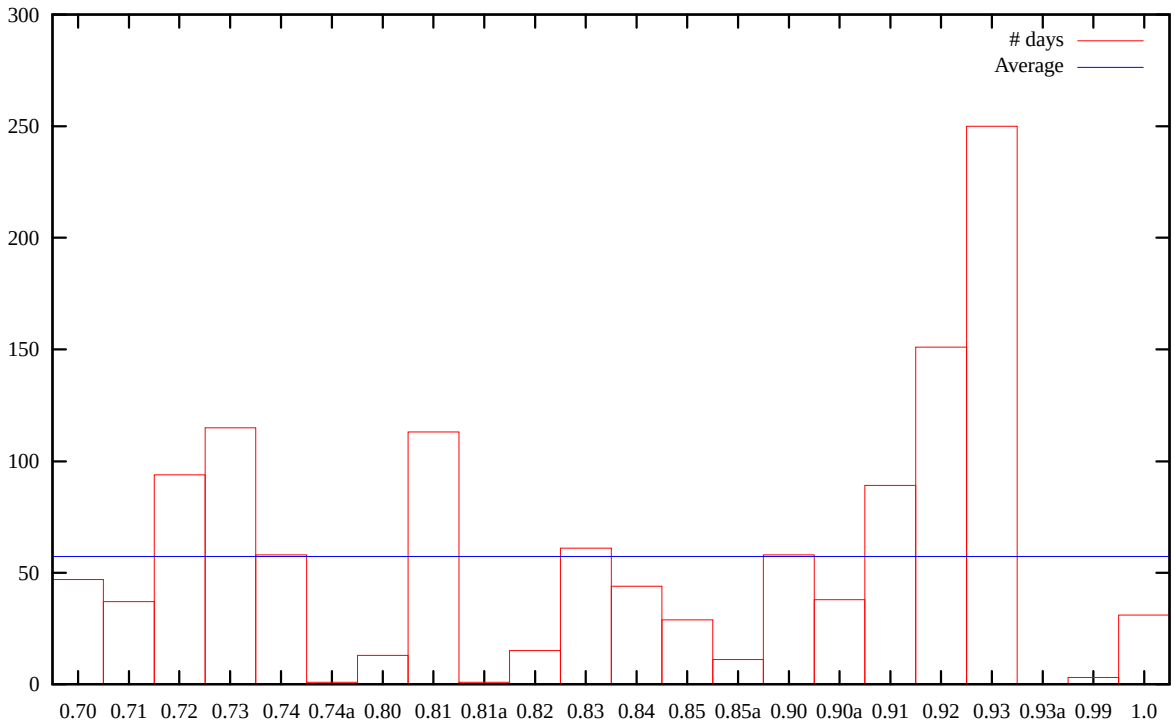

Figure 1: The number of development days per release.

2 Tar size per release

The size of a tar archive of the source tree gives a good indication of the amount of source code a project has. The amount of source code usually increases when features are added, but decreases when code cleanups are performed.

Table 2 shows the sizes of the source archives. The sizes are plotted in figure 2.

Version	Bytes	MiB
0.70	1413120	1.35
0.71	1454080	1.39
0.72	1474560	1.41
0.73	1515520	1.45
0.74	1515520	1.45
0.74a	1515520	1.45
0.80	1597440	1.52
0.81	1689600	1.61
0.81a	1689600	1.61
0.82	1689600	1.61
0.83	1710080	1.63
0.84	2099200	2.00
0.85	2078720	1.98
0.85a	2078720	1.98
0.90	1966080	1.88
0.90a	1966080	1.88
0.91	2078720	1.98
0.92	2099200	2.00
0.93	2150400	2.05
0.93a	2150400	2.05
0.99	2129920	2.03
1.0	2058240	1.96
Average	1823651	1.74

Table 2: The sizes of the tar archives.

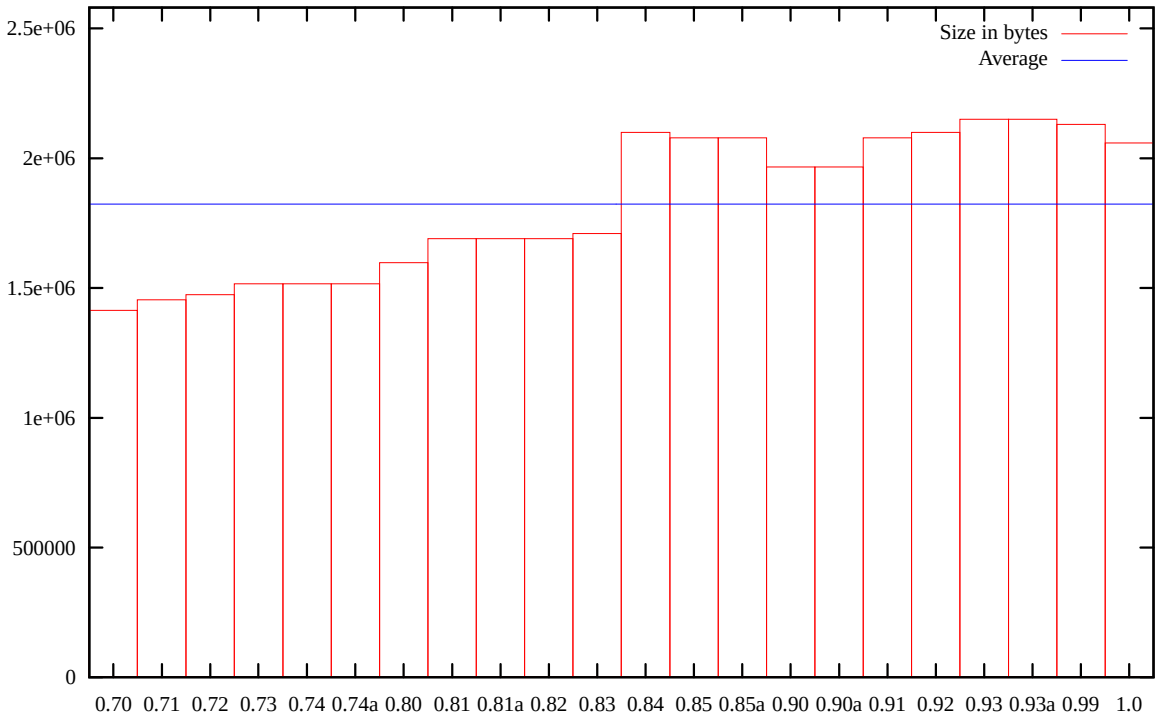

Figure 2: The sizes of the tar archives.

3 Compressed tar size per release

Users download the compressed tar archive when they want to compile (and presumably install) the software from source code. Table 3 shows the sizes and they are plotted in figure 3.

Version	Bytes	KiB
0.70	301563	294.50
0.71	316106	308.70
0.72	322185	314.63
0.73	333099	325.29
0.74	333185	325.38
0.74a	333311	325.50
0.80	351775	343.53
0.81	373036	364.29
0.81a	373025	364.28
0.82	375607	366.80
0.83	380672	371.75
0.84	473930	462.82
0.85	470233	459.21
0.85a	470425	459.40
0.90	433321	423.17
0.90a	433338	423.18
0.91	462675	451.83
0.92	464110	453.23
0.93	478298	467.09
0.93a	478306	467.10
0.99	469547	458.54
1.0	453937	443.30
Average	403713	394.25

Table 3: The sizes of the compressed tar archives.

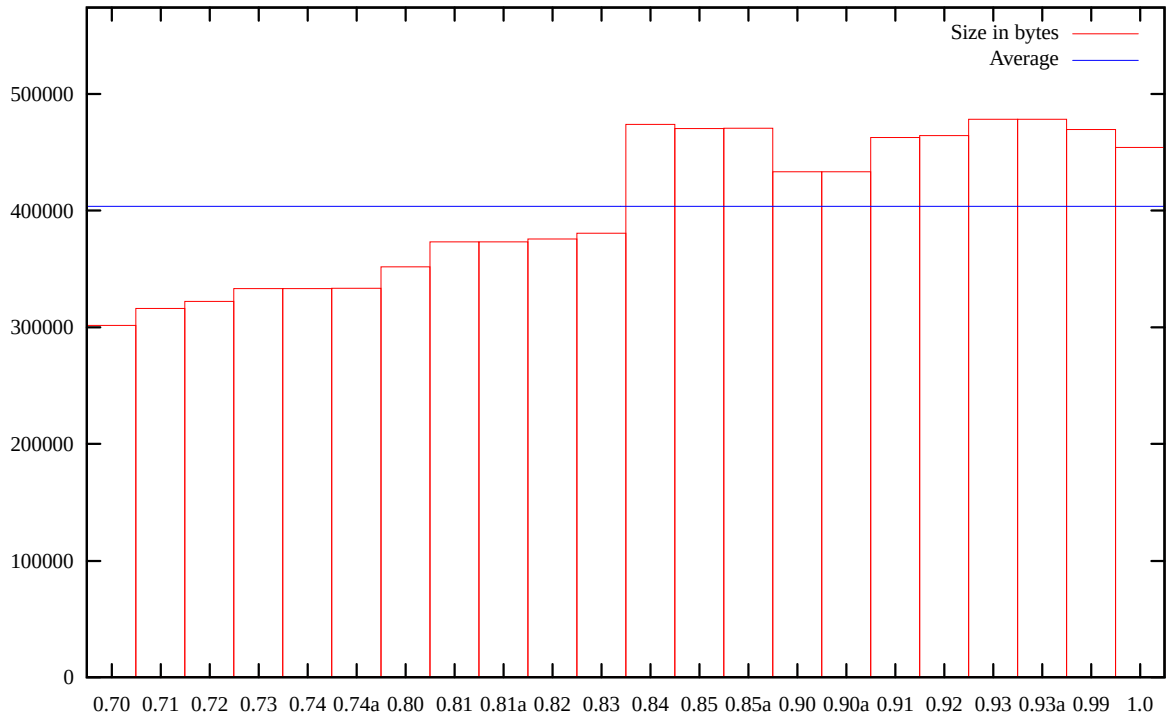

Figure 3: The sizes of the compressed tar archives.

4 Source lines of code per release

The amount of code written is an indication of the effort that has gone into a project. This section will show the amount of code in lines. Table 4 shows the amount in thousands of lines (Kloc) and the values are plotted in figure 4. The data is generated using David A. Wheeler's 'SLOCCount.'

Version	Kloc
0.70	30.06
0.71	30.41
0.72	30.89
0.73	32.04
0.74	32.04
0.74a	32.05
0.80	32.70
0.81	33.78
0.81a	33.78
0.82	33.83
0.83	34.12
0.84	42.46
0.85	43.51
0.85a	43.50
0.90	41.33
0.90a	41.34
0.91	41.23
0.92	41.23
0.93	41.50
0.93a	41.50
0.99	41.51
1.0	39.89
Average	37.03

Table 4: The amount of code per release in thousands of lines of code.

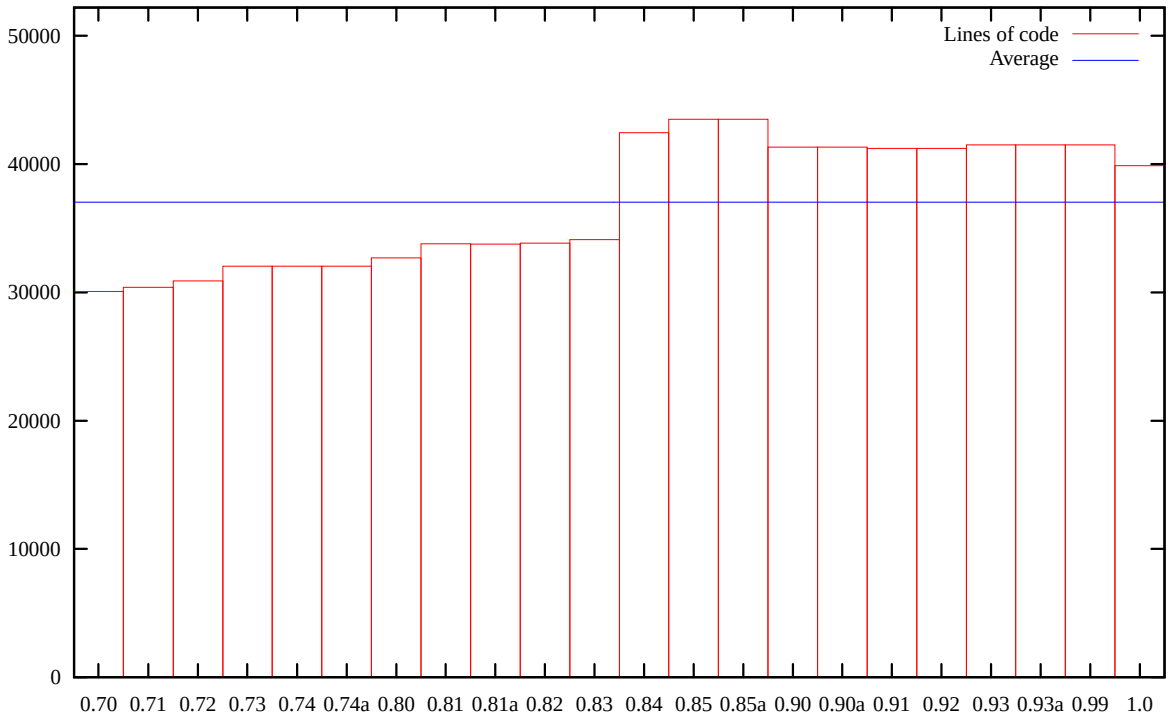

Figure 4: The amount of code per release in lines of code.

5 Development effort

A nice feature of SLOCcount is that it can estimate the development effort. For doing this it uses the COCOMO model—an absolutely useless model if you haven't got good estimates of what the constants should be for your organisation/project—but it gives a nice statistic to display. You can find the numbers in table 5 and the graph in figure 5. The data is, of course, generated using David A. Wheeler's 'SLOCCount.'

Version	Person years
0.70	7.13
0.71	7.21
0.72	7.33
0.73	7.62
0.74	7.62
0.74a	7.62
0.80	7.79
0.81	8.06
0.81a	8.06
0.82	8.07
0.83	8.14
0.84	10.24
0.85	10.51
0.85a	10.51
0.90	9.96
0.90a	9.96
0.91	9.93
0.92	9.93
0.93	10.00
0.93a	10.00
0.99	10.00
1.0	9.59
Average	8.88

Table 5: The amount of development effort in person years.

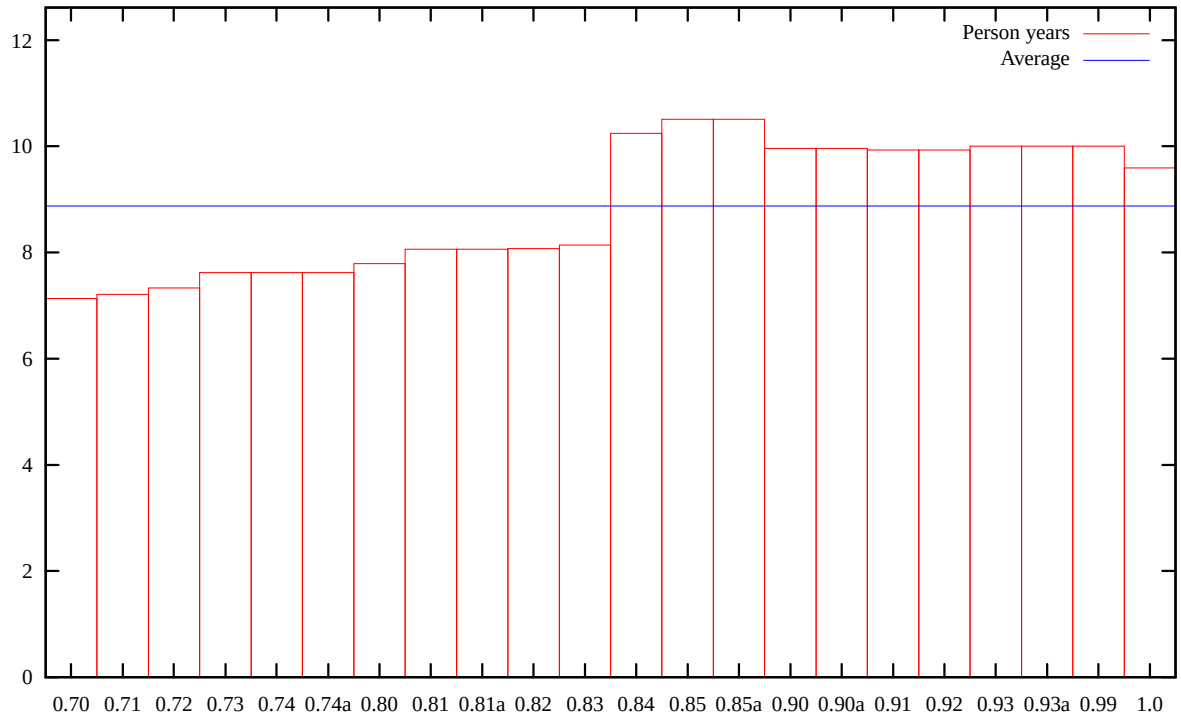

Figure 5: The amount of development effort in person years.

6 Development cost

The really interesting (but equally nonsense) figure generated by SLOCcount is the total cost to develop. It answers the question what it would've cost to develop the program in a company—and therefore what the software's worth. The figure is kind of interesting—mostly for your ego.

The data is generated using David A. Wheeler's 'SLOCCount' and can be found in table 6 and figure 6.

Version	**Cost ($\times$ \$ 1000)**
0.70	962.79
0.71	974.43
0.72	990.72
0.73	1029.52
0.74	1029.52
0.74a	1029.65
0.80	1051.73
0.81	1088.23
0.81a	1088.20
0.82	1090.09
0.83	1099.94
0.84	1383.70
0.85	1419.48
0.85a	1419.34
0.90	1345.10
0.90a	1345.23
0.91	1341.61
0.92	1341.64
0.93	1350.70
0.93a	1350.70
0.99	1351.11
1.0	1295.73
Average	1199.05

Table 6: The total development cost per release.

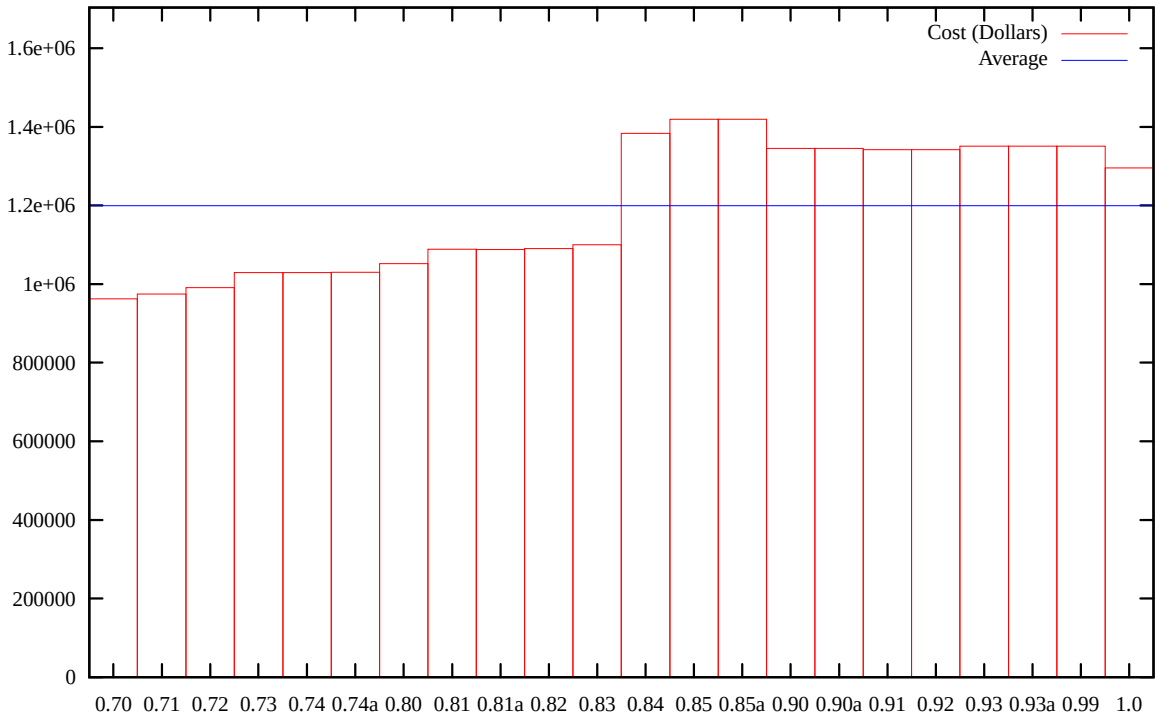

Figure 6: The total development cost per release.

7 Source lines of code against time

As shown in section 1, releases can take a variable amount of time to complete. In figure **??** I've plotted the size of the code base (in source lines of code) against time, instead of against versions.

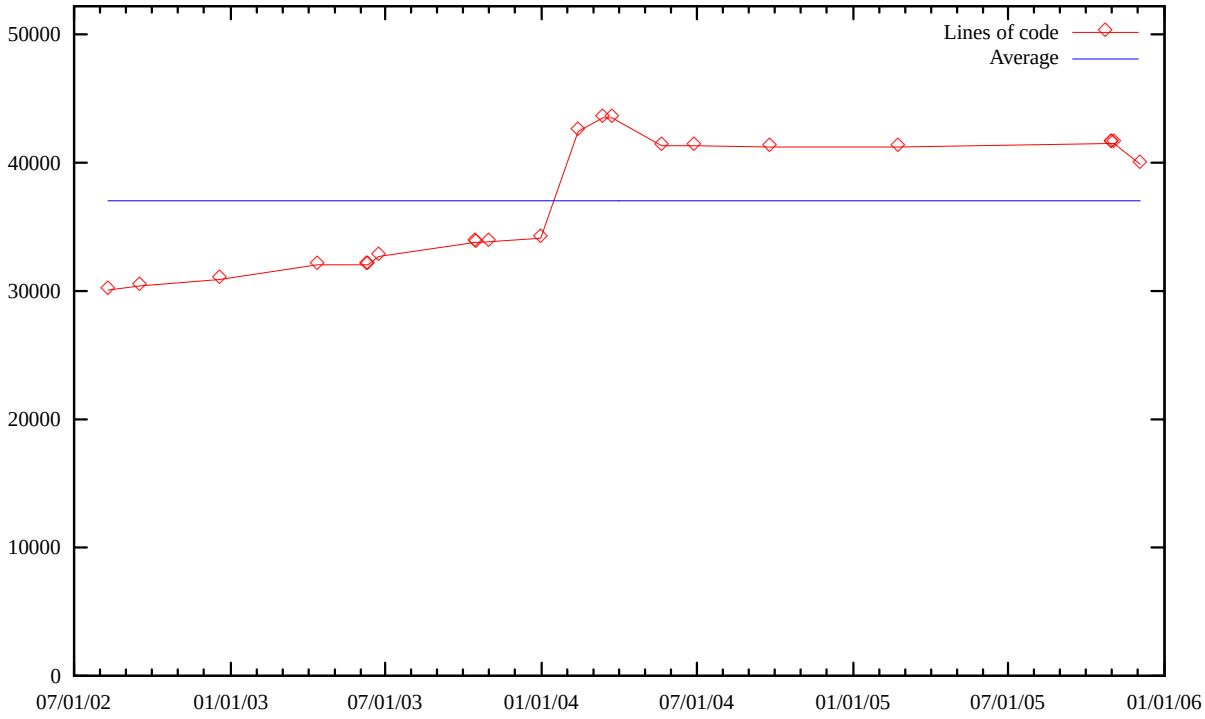

Figure 7: The amount of code in source lines of code plotted against time.