

Estimating source lines of code from object code: Windows and Embedded Control Systems

Les Hatton
CISM, University of Kingston*

August 3, 2005

Abstract

Much defect data is expressed in terms of defect density (defects per 1000 lines of code) however in many proprietary systems such as Windows and also in embedded control systems, only the number of bytes of object code is normally known making it difficult to compare such systems in terms of standard measures like asymptotic defect density. This paper describes simple methods of calculating equivalent lines of code to allow such comparisons to be made and quotes estimated lines of code for various versions of Windows. The calibrations have been done for C which still dominates operating system and embedded control systems implementations.

\$Date: 2005/05/25 16:12:07 \$

1 Overview

Although standard engineering reliability is quoted in such measures as MTBF (Mean Time Between Failures), MTTF (Mean Time to Fail), or Probability of Failure on Demand, such data is rarely available with software systems so defects per 1000 lines of code, a volumetric measure, has become a standard measure for software system quality over the last thirty years. Defect density is not entirely satisfactory owing to a degree of interpretation as to what constitutes a defect and particularly what constitutes a line of code, but in spite of this, it provides a reasonable and widely used scale of quality.

1.1 Defect density measurement

The scale of defect density used here is the asymptotic defect density in defects per 1000 executable lines of code. Executable lines give some degree of independence of programming language and the use of an asymptotic value places an upper bound on the total number of defects which will arise before use of a system is terminated, [2].

*L.Hatton@kingston.ac.uk, lesh@leshatton.org

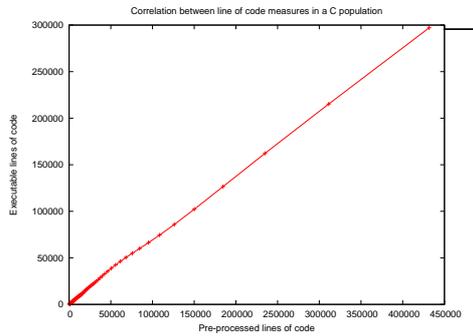


Figure 1: The relationship between pre-processed and executable lines for a large C population

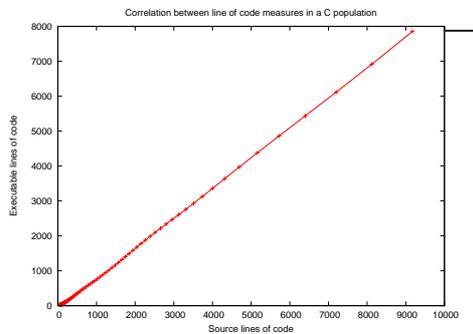


Figure 2: The relationship between source and executable lines for a smaller C population

A line of code Although executable line of code, (that is a line of code which directly generates object code) is used here, it is perfectly possible to use other measures provided they are correlated in some sense against executable lines. Such measures depend on the implementation programming language. For C programs as dominate embedded control systems, Figure 1 shows the relationship between the commonly used measure of pre-processed lines of code, (lines which have been through the C pre-processor removing comments and blank lines but expanding include files) and executable lines for the large C population studied in [1]. As can be seen, the relationship is very close and measurements of one can be used for accurate measurements of the other. To complete this analysis, the relationship between source lines and executable lines for a C population is shown in Figure 2

Some typical values of defect density By this standard, a value of around 0.1 represents amongst the lowest reliably achieved, for example as quoted for the NASA Shuttle software, [3], a value of between 0.1 and around 1.0 represents software of excellent quality, 1.0 - 4.0 represents software of reasonable commercial quality and a value of greater than 4.0 would not be very satisfactory, [2].

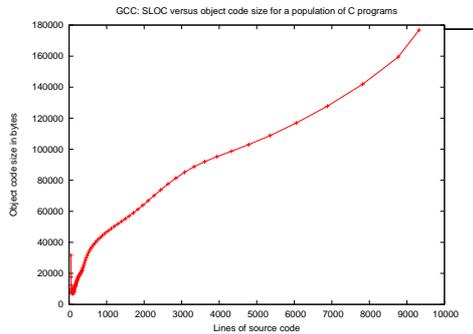


Figure 3: The relationship between source lines of code and object code for a C population compiled with the GNU gcc compiler

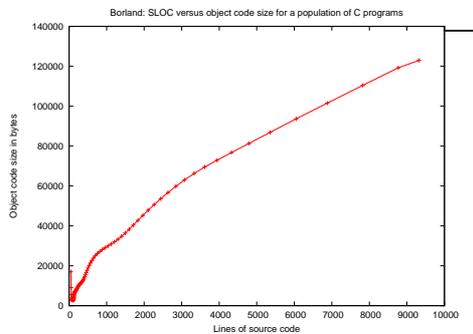


Figure 4: The relationship between source lines of code and object code for a C population compiled with the Borland C++ compiler

1.2 Source code and object code

This is clearly a function of the compiler and the optimisation level. The relationship between source code and object code for a different population of C systems for two different compilers is shown as Figure 3 (GNU gcc) and Figure 4 (Borland). As can be seen conversions are pretty consistent in both compilers with ranges of between around 14 and 20 bytes per source line being recorded depending on the compiler.

2 Conversion ratios

The above information provides a simple method for calculating equivalent lines of code. Table 1 summarises the conversion ratios. Error bounds are quoted to allow approximately for optimisation level and different compiler for the byte conversions and allow for small errors in the Figures shown above.

From/To	C Source lines	C Pre-processed lines	Executable lines	Bytes
C Source lines	1.0	1.24 ± 0.11	0.87 ± 0.03	17 ± 3.5
C Pre-processed lines	0.81 ± 0.07	1.0	0.7 ± 0.04	14 ± 4.0
Executable lines	1.15 ± 0.04	1.42 ± 0.08	1.0	19.5 ± 4.2
Bytes	0.058 ± 0.01	0.071 ± 0.16	0.051 ± 0.08	1.0

Table 1: Conversion ratios between various code measures

3 Estimating the number of lines of code in Windows operating systems

On Windows systems, the relevant object code is collected into two directories, Systems and Systems32. Using a source line conversion rate from Table 1 of 17.5 ± 3.5 , Table 2 gives estimates of the corresponding size of Windows in lines of code.

OS	Size in Object code (Mb)	Estimated size in source lines (MLOC)
Windows 98	174	9.9 ± 1.6
Windows 2000	539	30.8 ± 2.9
Windows XP	699	39.9 ± 4.9

Table 2: Estimating the number of lines of code in Windows systems

Given the quoted figure of 63,000 defects¹, this allows an estimate at the defect density of Windows 2000 to be made as being in the range $1.8 - 2.2$ per *KLOC* which is reasonable but not very good.

4 Conclusions

A simple method of estimating the number of code lines corresponding to a given object code size in bytes is presented and shown to give stable predictions which are similar to others which have been quoted, [4].

References

- [1] L. Hatton. *Safer C: Developing software in high-integrity and safety-critical systems*. McGraw-Hill, 1995. ISBN 0-07-707640-0.

¹<http://archives.cnn.com/2000/TECH/computing/02/17/win2k.bugs.idg>

- [2] L. Hatton. *Software Failure: avoiding the avoidable. The art and essence of Forensic Software Engineering*. Addison-Wesley, 2006. To be published.
- [3] T.W. Keller. Achieving error-free man-rated software. *Second International Software Testing, Analysis and Review Conference*, 1993. Monterey, USA.
- [4] Gary McGraw. From the ground up: The dimacs software security workshop. *IEEE Security and Privacy*, 1(2):59–66, March 2003.