C. Wohlin and M. C. Ohlsson, "Reading between the Lines: An Archival Study of Software from Nine Releases", Proceedings ICSE workshop on Software Change and Evolution", Los Angeles, USA, 1999.

Reading between the Lines: An Archival Study of Software from Nine Releases

Claes Wohlin and Magnus C. Ohlsson
Dept. of Communication Systems
Lund Institute of Technology, Lund University
Box 118, SE - 221 00 Lund, Sweden
E-mail: (claes.wohlin, magnus_c.ohlsson@tts.lth.se)

1. Introduction

Today, most software systems evolve over several releases. This implies a need to study and understand how it changes over time. Still, it is not common practice to collect metrics on a regular basis with the objective to track software change between software releases. Although, metrics may not have been collected with this objective in mind, it may in many cases be possible to perform an archival study [Robson93]. An archival study may provide “an after the fact” understanding of the software. Moreover, it may provide important information for future releases, and also a more general understanding of software evolution.

This paper presents an archival study of nine software releases. The study is primarily based on lines of code, which can be divided into comments, blank lines and non-comments and non-blanks (i.e. executable lines of code) and the number of functions in the software. The study includes a correlation study (Section 2), an evolution study, i.e. changes between releases (Section 3), an outlier analysis (Section 4), and a study of the percentage of different types of lines of code (Section 5). Finally, a brief discussion is provided in Section 6.

2. Measures

The study is based on lines of code measures and the number of functions in the code. The data has been obtained second-hand. In other words, first hand knowledge of the data is missing. Thus, the interpretation of the study is based on the data, and not on knowledge and experience of the actual system and projects. The code is written in C, but no other information is available due to confidentiality. Thus, the understanding has to be based only on the available lines of code measures. The following measures are used:

- Lines of code (LOC): a simple count of all lines,
- Comments: a count of the number of lines with comments,
- Blank lines: a count of the number of blank lines in the code,
- NCNB: a count of the number of executable lines of code,
- Functions: a count of the number of functions in the system releases.

Some of the measures are related as follows:

LOC = Comments + Blank lines + NCNB

The measures are highly correlated as can be seen from Table 1.

TABLE 1. Correlations between the measures.

Correlation	Functions	Comments	Blank lines	NCNB
Functions	1	0.9844	0.9981	0.9922
Comments	0.9844	1	0.9899	0.9965
Blank lines	0.9981	0.9899	1	0.9941
NCNB	0.9922	0.9965	0.9941	1

LOC is not included in the correlation study for obvious reasons, i.e. it is the sum of the other measures. The high correlation between the measures implies that the releases have maintained a similar relative relationship between the different types of lines of code as well as with the number of functions. This is further investigated below when studying the relative differences between the different types of lines of code in Section 5.

3. Evolution between releases

3.1 Increase in measures

The high correlations between the measures mean that it is sufficient to show the evolution in terms of software growth for one of the measures. This in combination with the relative differences in the following subsection provide a comprehensive picture of the software growth over the nine releases. The growth of lines of code (LOC) is shown in Figure 1.

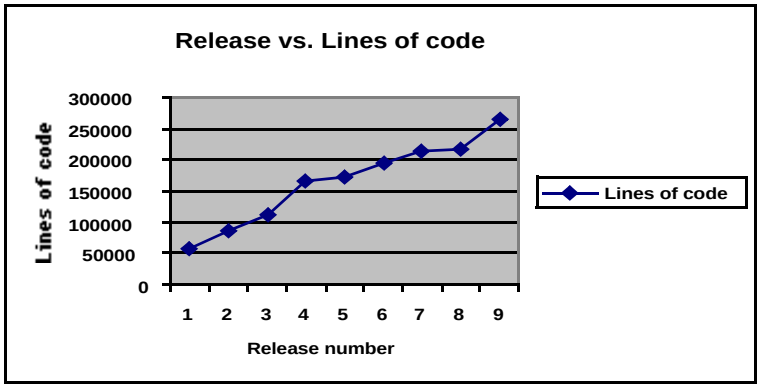

FIGURE 1. Increase in lines of code over the nine releases.

From Figure 1, it can be seen that the system has grown from 50 KLOC to a little over 250 KLOC in the nine releases. Moreover, it can be seen that the system has in particularly grown in the first four releases, and that the increases have been smaller except for the last release. This can also be seen in the following subsection.

3.2 Relative differences

The growth of the different measures over the nine releases can be seen from the bar chart in Figure 2.

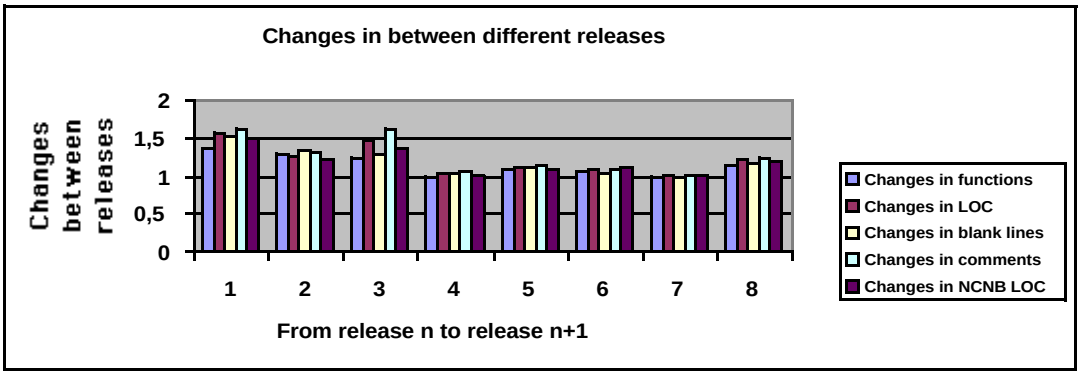

FIGURE 2. Increase of the different measures between the releases.

The bar chart provides the growth in terms of percentage between the releases, for example, the growth between release 1 and 2 are approximately 50%. Thus, number 1 on the x-axis indicates the growth from release 1 to 2. The bar chart supports the conclusions draws in the previous subsection.

4. Outlier analysis

The general understanding in the previous sections has to be complemented with an outlier analysis. An outlier analysis of the size of the functions is shown in the box plot in Figure 3.

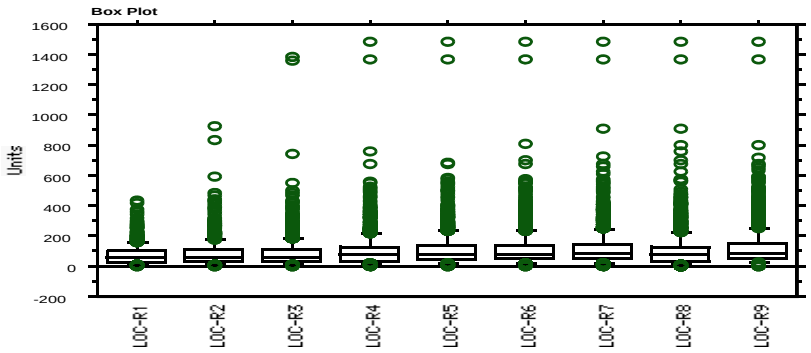

FIGURE 3. A box plot of the size of the functions in terms of LOC.

It can be seen that two outliers seem to be the same from release 4 and onwards. A closer look at the outliers show that they have a large amount of comments. Thus, the size as such may not be a problem as long as the comments are motivated by providing support for future maintenance rather than being a result of a large number of faults. The latter does not seem to be the case as the comments were all added when the new functions were introduced. If it would have been a result of corrective maintenance the functions are likely to change over at least a couple of releases before all problems with them have been removed. It should also be noted that the outliers from release 2 and 3 have become non-outliers from release 4 and onwards.

The outlier plot in Figure 3 should be complemented with mean values and standard deviations of the functions over the nine releases. The mean values and standard deviations of the lines of code (LOC) for the nine releases are shown in Table 2.

TABLE 2. Mean and standard deviation of the lines of code of the functions.

Function	R1	R2	R3	R4	R5	R6	R7	R8	R9
Mean	74	84	83	98	105	109	113	101	120
Stand. dev	66	83	93	102	104	106	110	111	112

As can be seen from Figure 3 and Table 2, the mean lengths of functions are rather low, but some functions are fairly large as can be seen from the box plot in Figure 3.

The outlier analysis of the size of the function should be complemented with an analysis of the functions which in some way have a different structure, for example, very few comments or very many comments in comparison with the number of executable lines of code. Few comments could indicate functions which will be difficult to maintain in the future due to poor documentation. Many comments may also be a problem, since it may be an indication of several problems. It is quite common that comments are written when faults are corrected, and hence many comments may be an indication of previous problems. This raises an important question, which is when measures are collected from the code in comparison with when faults are corrected. For example, a lot of comments and many faults cannot be interpreted as that the comments cause the faults, on the contrary the comments may be an indication of that we have had many faults previously. Thus, it is important to keep track of cause and effect.

In order to not only focus on the large functions, the outliers in terms of Comments/ NCNB are also investigated. The objective is to both identify functions with very few comments and functions which contain unexpectedly many comments. The analysis of functions with few comments include identification of functions with no comments and functions with few comments for each executable line. The number of functions with no comments are shown in Table 3 together with the total number of functions in each release.

TABLE 3. Functions without comments and the total number of functions in each release.

No com.	R1	R2	R3	R4	R5	R6	R7	R8	R9
Number	135	159	175	164	168	163	166	168	168
Total	763	1043	1339	1676	1650	1787	1888	1896	2189

Two functions are pin-pointed when performing an outlier analysis looking for functions with very few comments. Both functions were introduced in the first release and they have not been changed since the first release. Thus, the functions seem to be fairly stable. Based on the information from Table 3, and from the fact that the two outliers were introduced in the first release, it is likely that comments were regarded as important and used to a larger extent after the first release. This is also shown below, when studying the distribution of the different types of lines of code in Section 5.

The functions with disproportionately large number of comments are a few functions introduced in release 6 and 7. These functions contain only a few number of executable lines of code, and it is likely that these functions contain important descriptive comments. Thus, the functions are probably not a problem.

5. Distribution of the different types of lines

To further increase the understanding of the software, the distribution of the different types of lines is studied. The distribution of the different types is shown in Figure 4.

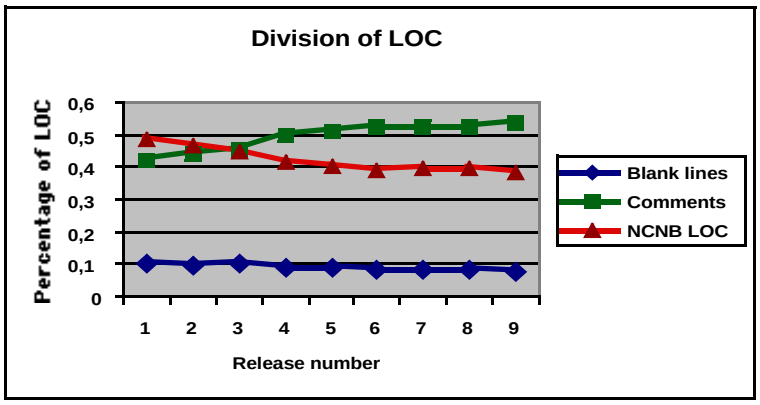

FIGURE 4. Distribution of the different types of lines of code.

From Figure 4, it can be seen that the amount of comments is large throughout the evolution of the software. The number of comments was fewer in the first releases, but as also has been shown above the number of comments increased as the system grew larger. The number of blank lines is fairly stable over the nine releases.

6. Discussion

This paper has reported of an archival study of nine consecutive software releases. The objective has been to try to understand as much as possible from just investigating the software code. The study has raised several relevant questions in order to keep track of the evolution and enable prediction of future releases. Important questions include:

- What can we learn from archival studies of code?
- What type of measures should we collect to keep track of software evolution?
- What can we predict using historical data from previous releases?
- How do we determine cause and effect, for example, complexity vs. faults? Have we measured the complexity prior to the changes due to faults? Is the complexity a result of the faults?

The objective of the study was to see what can be learnt from an archival study of software code. It is clear that some lessons can be learned from studying the code, but it is not enough. Metrics have to be collected using a goal-oriented approach to enable full control of the evolution, and also to support prediction of, for example, critical components from a maintenance perspective [Ohlsson98].

References

[Ohlsson98] M.Ohlsson and C. Wohlin, “Identification of Green, Yellow and Red Legacy Components”, Proceedings of International Conference on Software Maintenance, pp. 6-15, 1998.

[Robson93] C. Robson, “Real World Research”, Blackwell Publishers,1993.