Analysis of the Linux Kernel Evolution Using Code Clone Coverage

Simone Livieri† Yoshiki Higo† Makoto Matsushita† Katsuro Inoue†

†Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: {simone, y-higo, matusita, inoue}@ist.osaka-u.ac.jp

Abstract

Most studies of the evolution of software systems are based on the comparison of simple software metrics. In this paper, we present our preliminary investigation of the evolution of the Linux kernel using code-clone analysis and the code-clone coverage metrics. We examined 136 versions of the stable Linux kernel using a distributed extension of the code clone detection tool CCFinder. *The result is shown as a heat map.*

1. Introduction

Evolution is essential and indispensable for a software system *"else it will become progressively less satisfactory"*[7]. Usually, software evolution is understood and characterized by changes in the version number, number of lines of code, the release date, or other simple metrics.

Code-clone analysis is a good vehicle to quantitatively understand the differences and improvements between two versions of the same software system [4, 6]. It is more precise than simple string-matching analysis for what concerns identifying code changes in the case of renaming of files or code relocation; therefore, it's possible to characterize the evolution of a software system using code-clone analysis.

However, code-clone analysis is generally resource expensive and there is a limitation in its application to large source code repositories. In order to break this limitation, we have developed a distributed system for code-clone analysis, named D-CCFinder[8].

We have analyzed the evolution of the Linux kernel using D-CCFinder, computed the change ratios between different versions, and visualized the results using a heat map graph, in which each square represent the value of the code clone coverage ratio (see Section 3) between versions of the Linux kernel, and its width and height are proportional to the size of the kernels.

Similar studies have been conducted in the past: Van Rysselberghe *et al.* reconstructed the evolution of a software system using code-clone detection techniques[9]; Godfrey *et al.* have conducted a thorough investigation of the evolution of the Linux kernel measuring various characteristics of the source code[2]; more recently, a similar study has been conducted by Izurieta *et al.*[5].

However, no extensive investigation of the software system evolution using large-scale code clone analysis has been performed. In this paper, we will see the code clone coverage between any two versions of 136 stable Linux kernels.

In Section 2, we will describe these target versions, and in Section 3 we will show our approach. In Section 4 the results and discussion are presented, and in Section 5 we will conclude our discussions with some future remarks.

2. The Linux Kernel

The Linux Kernel was created 15 years ago by Linus Torvalds, and the first stable version (1.0) was released in March 1994. This release contained 313 `.c` files for a total of 141,388 lines of code with a size of 3.8 MBytes.

The Linux kernel is maintained in two versions: *stable* and *development*. By convention, the middle number in a kernel version identifies the type of kernel: stable releases are identified by even numbers (*e.g.* 2.0.5) while releases in the development branch are identified by odd numbers (*e.g.* 2.3.32).

At the time of writing, the most recent stable kernel is version 2.6.18.3. It contains 8415 files, 5,476k line of codes (including comment and blank lines) and has a size of 157 MBytes.

This study examined 136 versions of the stable kernel as shown in Table 3. Only the `.c` files were considered, the size of each release was measured using the Unix command `du -k`, and the number of lines of code was counted using the Unix command `wc -l`.

Fourth International Workshop on Mining Software Repositories (MSR'07)
0-7695-2950-X/07 $20.00 © 2007 IEEE

3. Methodology

For each pair (V_i, V_j) of kernel versions we computed the code clone coverage ratio $Coverage_{ij}(i,j)$ defined as follows:

$$Coverage_{ij}(i,j) = \frac{Loc(C_{ij})}{Loc(V_i) + Loc(V_j)}$$

with:

C_{ij}: code clone segments between V_i and V_j;
$Loc(x)$: the total number of lines of code in x.

We have also measured some aspects of the growth of Linux in order to validate the findings from the code clone analysis.

The analysis was performed by running D-CCFinder and all associated tools, with a detectable minimum token length of 50 tokens. Figure 4 shows the resulting heat map. Because code-clone coverage of each kernel version against itself is obviously 100%, we didn't compute it. The heat map has been generated by normalizing the code clone coverage values to 1.0.

The maximum code clone coverage has been measured as 67%; a value close to 100% would be expected but two factors contributed to decreasing it:

- in order to remove redundancy and uninteresting clones consisting in simple repeated patterns, the Repeated Token Ratio (RNR) metrics has been used to perform a filtering of the code clone data[3] (in our experiments we set the threshold value for RNR to 0.5);
- because of the method used, the reported count of line of codes is greater than the number of non-commentary lines of code.

4. Results and Observations

In this section we analyze and present the results of the examination of the 136 different kernel versions. This analysis is a preliminary research that we plan to further conduct and integrate with new findings.

- The heat map in Figure 4 shows, as expected, the highest code coverage ratios near the diagonal: because changes between consecutive releases are usually small, the code clone coverage is high.
- The first observation is that, for major versions 2.0, 2.2 and 2.4 the color pattern of the triangular sector is the same. Interpreting this pattern, we can infer a considerable and steep growth of the size of the kernel during its life cycle.

Version	Loc	Size (KBytes)	Number of versions
1.0	141K	3926	1
1.2.0∼	234K	6534	14
1.2.13	238K	6596	
2.0.0∼	563K	16076	41
2.0.40	768K	21952	
2.2.0∼	1310K	37056	27
2.2.26	1970K	58812	
2.4.0∼	2366K	69200	34
2.4.33.4	3865K	112148	
2.6.0∼	4120K	120030	19
2.6.18.3	5476K	157290	
Total number of versions			136
Number of .c files			376,596
Total lines of code			266,943,565
Total size			7.4 GBytes

Table 1. Characteristics of the analyzed kernel versions.

The growth of the kernel is confirmed by Figure 4, showing the number of lines of code for each of the examined versions of the Linux kernel, normalized respect to the size of the most recent release (at the time of writing) of its major version (*e.g.* version 1.2.4 has been normalized respect to version 1.2.13). It is easy to see how the kernel size changed greatly between version 2.0.29 and 2.0.37, version 2.2.10 and 2.2.19, and version 2.4.9 and 2.4.25. A closer investigation of the source code has shown that these changes are mostly due to the introduction of new hardware drivers, supported architectures, and features (network protocols, file system drivers, etc.). We assume that most of these additions were "back-ported" from the parallel development branches, and for some of them our assumption has been proven right by the developers' comment in the source code. Figure 3 shows the growth rate of the `arch`, `drivers`, `fs` and `net` subsystems compared with the growth of the whole source tree for each version of the kernel.

- The next interesting fact is that from the observation of Figure 4 it is easy to discern the development path of the various versions. Let's consider, for example, the area relative to the major version 2.4.

  The green part (`A`) represents the code-clone coverage for the kernel versions from 2.4.0 to 2.4.21 and from 2.4.22 to 2.4.40: the value is relatively low and it indicates a low degree of similarity among these versions.

  The upper and lower red-orange triangle (`B` and `C`) represents the code-clone coverage for the kernel versions

Fourth International Workshop on Mining Software Repositories (MSR'07)
0-7695-2950-X/07 $20.00 © 2007 IEEE

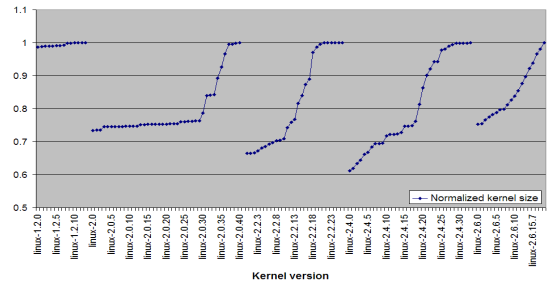

Figure 2. Normalized growth in the number of lines of code measured using the Unix command `wc -l` .

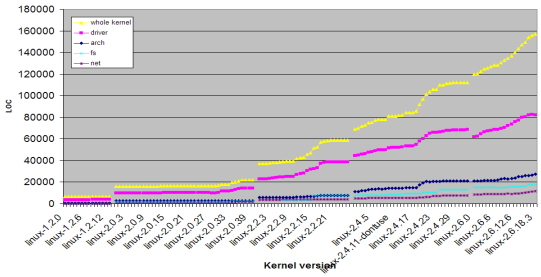

Figure 3. Growth of the `arch`, `drivers`, `fs` and `net` subsystems compared with the growth of the kernel source tree.

from 2.4.0 to 2.4.21, and from 2.4.22 to 2.4.40 respectively. The value is high and it indicates that these versions don't differ very much in size and composition. The zones with a more yellow color in `B` can be accounted to small changes that were made in the first releases (see Figure 4).

- We noted that the number of code clones in the Linux kernel is somewhat proportional to the size of the source tree. This can be partially explained considering that the Linux kernel can be divided in ten main subsystems, of which the fastest growing one is the `driver` subsystem[2]. Every driver must implement a uniform interface conforming to its hardware category (*e.g.* a driver for a network card must implement functions for sending and receiving packets), and, while drivers generally tend to be relatively self contained, drivers for the same family of hardware are likely to be very similar, often being implemented modifying already tested code. It is likely, then, that adding new drivers also adds new code clones.

- Versions 2.0, 2.2 and 2.4 present a similar evolutionary pattern, different from that of kernel 2.6, and has a code-clone coverage ratio near the diagonal distinctly higher than that exhibited by kernel 2.6. The main reason of these differences is the absence of a parallel development branch for technology testing and bug fixes: the new features and fixes are directly added to the main source tree. This is in contrast with what was done in the previous releases: new features and bug fixes were integrated in the stable branch after a period of incubation and testing in the development branch. Therefore contiguous versions of kernel 2.6 are relatively more different than contiguous versions of kernels 2.0, 2.2 and 2.4.

5 Conclusions

We have shown our preliminary results of the evolution analysis of the Linux kernel using code-clone based metrics. They might not be surprising, in the sense that most of them could be expected from the history of Linux[1]. However, by using code clone analysis, we were able to get a quantitative measurement of the evolution, and the metric values have been visualized. The nature of the evolution can be intuitively understood with the help of this visualization.

In order to scale the code clone analysis, we have used D-CCFinder which works on a 80 computers cluster in our student lab. We are using this system for the analysis of various targets such as different open source systems or proprietary applications.

We offered a coarse overview of the evolution of the various kernel versions using an heat map. In addition to this, we believe essential to investigate the details of the obtained clones, to precisely identify the characteristics of the code changes. We are going to elaborate the analysis presented in this paper to address this point.

Acknowledgments

This work has been conducted as a part of EASE Project, Comprehensive Development of e-Society Foundation Software Program, and Grant-in-Aid for Exploratory Research(186500006), both supported by Ministry of Education, Culture, Sports, Science and Technology of Japan. Also it has been performed under Grant-in-Aid for Scientific Research (A)(17200001) supported by Japan Society for the Promotion of Science.

References

[1] The linux kernel archives. `http://www.kernel.org`.

Fourth International Workshop on Mining Software Repositories (MSR'07)
0-7695-2950-X/07 $20.00 © 2007 IEEE

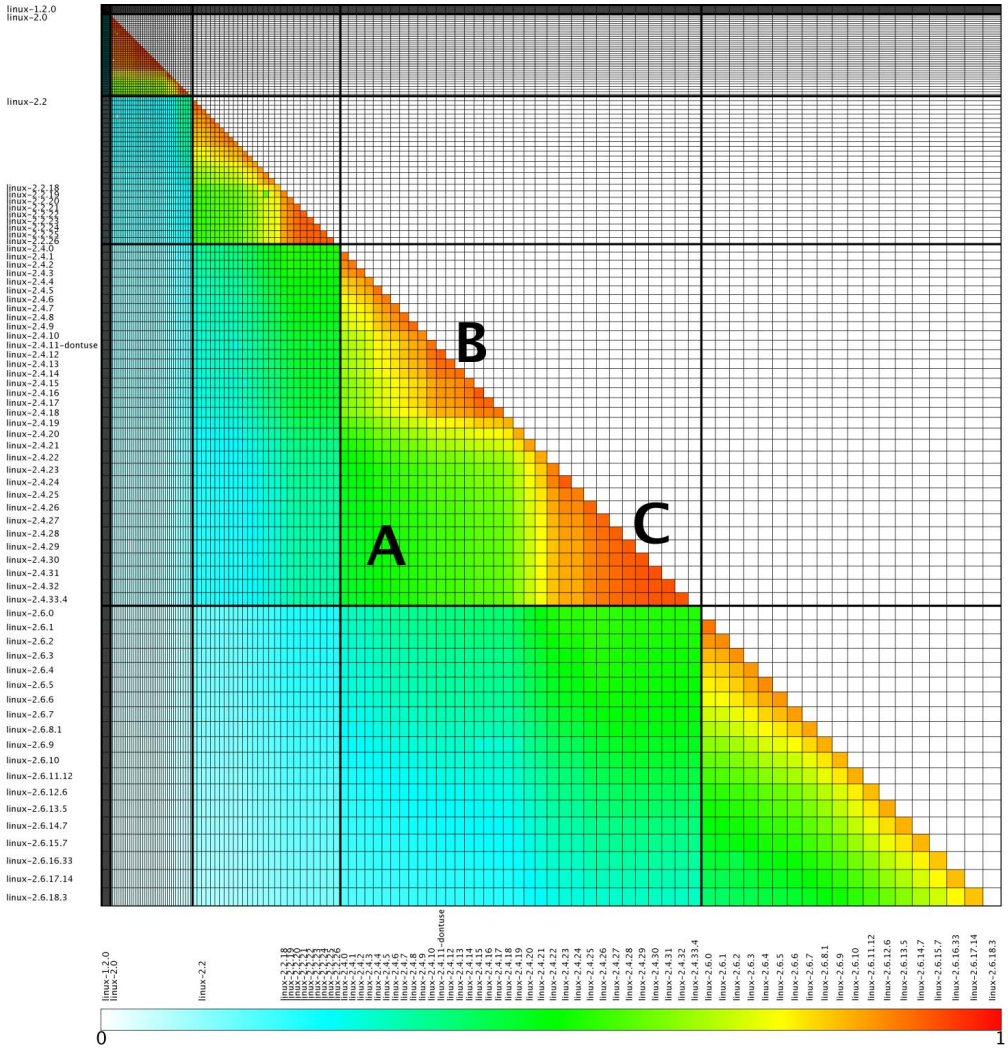

Figure 1. Heat map of the code clone coverage ratio between different Linux kernel versions.

[2] M. W. Godfrey and Q. Tu. Evolution in opensource software: A case study. In *Proc. of the 2000 International Conference on Software Maintenance*, pages 131–142, 2000.

[3] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue. Method and implementation for investigating code clones in a software system. *Information and Software Technology*, February 2007. To appear.

[4] Y. Higo, N. Yoshida, T. Kamiya, S. Kusumoto, and K. Inoue. Code clone analysis tool: ICCA. In *Proceedings of the Second Internation Workshop on Biologically Inspired Approaches to Advanced Information (Bio-ADIT 2006)*, 2006.

[5] C. Izurieta and J. Bieman. The evolution of freebsd and linux. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*, pages 204–211, New York, NY, USA, 2006. ACM Press.

[6] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, July 2002.

[7] M. Lehman, D. Perry, and J. Ramil. Implications of evolution metrics on software maintenance. In *ICSM '98: Proceedings of the International Conference on Software Maintenance*, page 208, 1998.

[8] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder. In *Proc. of the 2007 International Conference on Software Engineering*, Minneapolis, MN, USA, 2007. To appear.

[9] F. V. Rysselberghe and S. Demeyer. Reconstruction of successful software evolution using clone detection. In *Proceedings of the Sixth International Workshop on Principles of Software Evolution*, pages 126–130, 2003.

Fourth International Workshop on Mining Software Repositories (MSR'07)
0-7695-2950-X/07 $20.00 © 2007 IEEE