Yet Another Linux Kernel Case Study

Chris J Arges
University of Texas
Austin,Texas,USA
christopherarges@gmail.com

ABSTRACT

Source code metrics provide valuable data for improving software development as a whole. Calculating metrics on a wide and diverse set of data allows for a better understanding of how to apply these metrics to solve software engineering problems. Open source software provides a treasure trove of data waiting to be analyzed, and thus becomes an easy target to apply and use these metrics. In this paper, various versions of the Linux kernel are downloaded and analyzed for SLOC count to formulate hypothesis and ideas of future work. This exploratory research shows these results and compares findings with other research.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

Reliability, Design

Keywords

Metrics,Open Source

1. INTRODUCTION

The goal of this paper is to examine how the Linux kernel has grown over most of its twenty years of development. By measuring SLOC over time, distinct phases and paradigms can be shown that may have been the result of implicit or explicit design decisions by the community. In examining this metric, a view of growth and development of the Linux kernel becomes a bit clearer. In this paper, 557 releases of the Linux kernel from version 1.0 to version 3.0 have been downloaded and analyzed. Using an open source tool called SLOCCount [2] counts of each sub-directory and overall totals were calculated for each release.

First, previous research is examined showing how other papers have approached case studies of the Linux kernel. Next, I will examine some research questions guiding this paper. The Linux kernel as a project will be examined to gain a better understanding of its organization and history. The methodology used in this case study will be exposed. An evaluation section will show the results of this research and showcase any relevant findings. Finally, conclusions from these results will be explained as well as any ideas for future work.

2. PREVIOUS RESEARCH

There have been many other case studies performed on the Linux kernel already. Here a few papers are examined on their approach for showing SLOC metrics. In addition a summary and any conclusions reached are presented here.

2.1 "Estimating Linux's Size" [6]

This paper written by David Wheeler in 2000 shows the SLOC count for a complete Linux distribution (Red Hat 6.2). This means the source to every user application, library, as well as the Linux kernel were analyzed. Since Red Hat is an open source distribution, each source file can be easily downloaded and analyzed. While other studies were looking for changes over time, this study only focused on a single distribution release. The goal of this paper was to estimate the total cost to develop this entire suite of software given traditional software development models. In order to accomplish this, the author wrote a piece of software called SLOCcount. The results in 2000 estimated that it would cost $600 million dollars to develop in the U.S. by conventional means. [6]

2.2 "Evolution and Growth in Large Libre Software Projects" [5]

This case study is primarily concerned with showing how large open source projects grow over time. In particular, looking at the evolution and how these apply to Lehman's laws. [4] This paper examined the Linux kernel, various BSD kernels, as well as other larger open source projects. This project also used the tool SLOCcount [2] (developed by David Wheeler) to count SLOC in each of the respective projects. While source code repositories exist for the Linux kernel, the authors decided to download release archives of the source code and performed analysis on these. The authors examined Linux kernel versions 1.0 to version 2.6.10 which was the latest of the writing of the paper. The SLOC count for each release was plotted against time and graphed. In addition the authors tried to fit linear equations to the growth to refute Lehman's fourth law. The conclusion was that Linux kernel exhibits super-linear growth patterns. [5]

2.3 "The Linux Kernel as a Case Study in Software Evolution" [3]

This paper examines Linux kernel sources over a period of 14 years (1994-2008) to show how the system evolves over time. In particular how Lehman's laws [4] apply to the Linux kernel. Not only does it plot SLOC against release date, but

also other metrics such as McCabe's cyclomatic complexity, number of system calls, number of configuration options, and others. In addition to simply plotting these dependent variables versus time, the complexities per function, and CDF's of the complexity per function were examined providing greater insight. This paper did show a great deal of various metrics and overall is an exemplary case study of the Linux kernel.

3. RESEARCH QUESTIONS

The big goal of this research is to better understand Linux kernel software development in order to determine areas of improvement. For instance, it may be useful to examine a host of metrics and bug data over time to determine which metrics are best indicators of future bugs. This is important because many companies work with the Linux community to support new products, and reduction in debugging will reduce costs. In addition, by proposing a worthwhile model of collecting information and finding bugs this technique could be further studied and used in many other projects.

Since a complete study of this nature is beyond the scope of this paper, the focus will be mostly exploratory. Thus, the sub goal will be to understand development growth over time and how the various version schemes and sub-directories have grown over the years.

To achieve this goal the following will be answered: how has the source code changed from version 1.0 to version 3.0?

To answer this question the SLOC (Source Lines of Code) in the Linux kernel will be examined with respect to time. In addition, the changes in kernel series numbers will be examined as well as the various sub-directories and architectures. Finally, conclusions about these findings will be made.

4. THE LINUX KERNEL

In order to understand some of the results of this case study, one must understand more about the Linux kernel. The Linux kernel is an Unix operating system clone started by Linus Torvalds in 1991. It is currently licensed under the GNU General Public License version 2, and is considered "open source" meaning its source is freely distributed.

Development occurs across the globe and by both volunteers and employed developers. Typically a developer creates a 'patch' or file that contains the differences between the original source and the new source and sends this to the appropriate mailing list and maintainer. This could be the Linux Kernel Mailing List (LKML) or other mailing lists dealing with various subsystems of the kernel. Once this patch has been received by the list, various comments can be posted about the patch giving feedback and thus the original developer can refine their patch. Once the patch is deemed worthy of inclusion, the maintainer will include it into their own repository. The maintainer repositories then get pulled into other trees and eventually into Linus' tree. At this point, it is considered to be in the mainline kernel and any releases will be taken from this tree.

Versioning of the Linux kernel has changed over the years and it is critical to understand this in reading the results in this paper. Version 0.1 are pre-releases and are really the initial code that Linus wrote. In this paper we do not consider these versions. Version 1.0 through 1.3 were the initial releases and contain much of the initial development. Version 2.0 marked a change in the versioning system, where even minor numbers (2.0, 2.2, 2.4, 2.6) marked stable versions, whereas odd minor numbers (2.1, 2.3, 2.5) marked development versions. However, as 2.6 progressed, the development version was dropped, and was included within the 2.6 line as well. Thus, experimental or developmental branches would be maintained elsewhere. The 2.6 line continued for about 7 years with the 2.6.x.y scheme, where the x and y portions of the version number changed. Finally at the 20th anniversary of the project, 3.0 was designated as the new kernel version.

The sub-directories in the Linux kernel have changed over the years, but some prominent ones are worth noting. "Arch" contains all architecture specific code to allow Linux to run on various hardware platforms. It is interesting to see the progression and support of new platforms as the project progresses. Another major directory is the "drivers" directory. This contains all driver module code for various devices that can be connected to a CPU. This includes sound, video drivers, input devices, hard drives and other peripherals. This constitutes a major bulk of the code since new devices are continually being created, and thus new drivers being needed for some devices. The following directories are contained in the 3.0 version and have been considered for this study: drivers, arch, fs, sound, net, include, kernel, mm, tools, crypto, security, scripts, lib, block, Documentation, ipc, virt, init, firmware, and samples.

Finally, the bulk of the code in the Linux kernel is written in ANSI C. This allows the kernel to be compiled on many platforms and allows for portability. In addition, other languages are used including: architecture specific assembly, bash, perl, and awk. Each kernel can be setup to handle various hardware configurations using CONFIG options when building the kernel. This allows multiple types of kernels to be built and optimized for particular platforms. The following platforms are supported in version 3.0 and have been considered for this study: arm, (powerpc|ppc), (x86|i386), (mips|mips64), m68k, (sparc|sparc64), sh, ia64, blackfin, cris, s390, alpha, parisc, tile, um, mn10300, frv, avr32, m32r, xtensa, microblaze, unicore32, h8300. [1]

5. METHODOLOGY

For this study 557 versions of the Linux kernel were downloaded from version 1.0 to version 3.0. These release archives were downloaded from a public mirror of `kernel.org`. Originally, it was planned to use the kernel git tree to extract even more information; however, this proved to be to slow when switching to various versions, and did not include earlier version of the kernel. This downloading was accomplished with a Bash script utilizing curl. Each download was constructed as to retain the original modification date as seen from the source of the download.

The SLOCCount [2] program was used to evaluate SLOC for each Linux release. This program can be found at: `http://www.dwheeler.com/sloccount/`, and is released under the GPL license. This program defines a Physical SLOC (source line of code) as: "a line ending in a newline or end-of-file

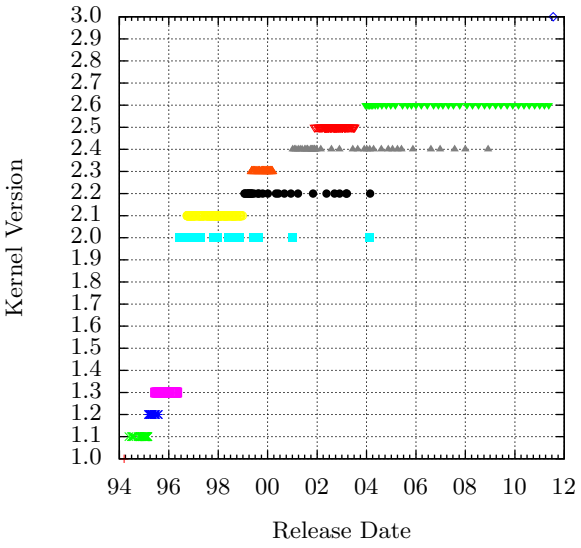

Figure 1: Linux Kernel Versions from 1.0 to 3.0

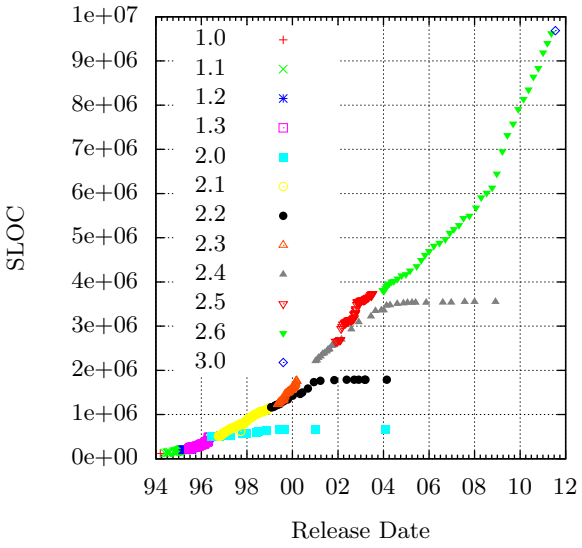

Figure 2: Total SLOC for Linux Kernel 1.0 to 3.0

marker, and which contains at least one non-whitespace non-comment character." [2]

Next, the bash script extracted each tarball and applied the SLOCcount program to produce a report showing lines of code per directory and total. In addition, an analysis of the arch sub-directory was also produced and output. The modification date of the file was extracted and output to this same file so that it could be used to provide an accurate date of when the archive was released.

After calculating the SLOC metric for each release, the output was parsed and formatted into tabular text data that could be interpreted by gnuplot and then graphed. Next this file was used with various gnuplot commands to produce a series of plots that are shown in the evaluation section.

6. EVALUATION

In this section a series of graphs show relevant sections of the data collected using SLOCCount against 557 versions of the Linux kernel. First, the version numbers are shown to illustrate the time-line in which these versions were released. Second, a study of the total SLOC for releases is shown to illustrate the overall course of development. Next, major directories are highlighted and their growth is shown in various ranges. Finally, the architecture sub-directory is explored to show how various architecture support came to be and how it affected the SLOC count of the project.

6.1 Version

Figure 6.1 shows the progression of the Linux kernel versions from 1.0 to 3.0. Here we see the intial 1.X series occur from March of 1994 until May of 1996. The 2.X series occurs from July 1996 until May 2011. The 2.X series prior to 2.6 has development versions (2.1, 2.3, 2.5) and stable versions (2.0, 2.2, 2.4). This graph shows that the stable versions have a longer 'tail' because they continue to be supported even as newer versions are created. The 2.6 version changed this, yet we can still see the 2.4 version was supported until December of 2008. Since there has only been one 3.0 release as of this writing, there only exists one data point in the corner showing this new version.

6.2 Total

Figure 6.2 shows the total SLOC for each Linux kernel release from 1.0 to 3.0 from March of 1994 until July of 2011. Each version (1.0, 1.1, 1.2, etc) has been color coded and the progression of each version can be shown on this graph. We can see some longer tails from the stable 2.X versions of the kernel (2.0, 2.2, 2.4) which were supported long after newer versions were created. This means that those versions had bug fixes 'back-ported' to the relevant version. This kept the kernel stable and at the same time had the benefit of bug fixes but without newer features. This graph shows that growth has not slowed down and in fact the addition of code into the kernel seems to have increased within the last 3 years. The 3.0 version the kernel has a total SLOC of 9,685,542. It would seem safe to predict that the SLOC of the Linux kernel will break 10 million very soon. Factors for these types of increases are not as apparent looking at such a high level, and thus it is important to look at growth of sub-directories.

6.3 Directories

Figure 6.3 shows the seven largest sub-directories by SLOC count in versions 1.0 to 3.0. Here we clearly see that the 'drivers' sub-directory is the largest and has been growing as quickly as the total SLOC. This intuitively makes sense because many new drivers are developed by many different companies and the code is fairly orthogonal. Next, in terms of size is the 'arch' directory which contains the architecture dependent code. As new platforms are being supported, this directory grows. This graph shows that the arch has been

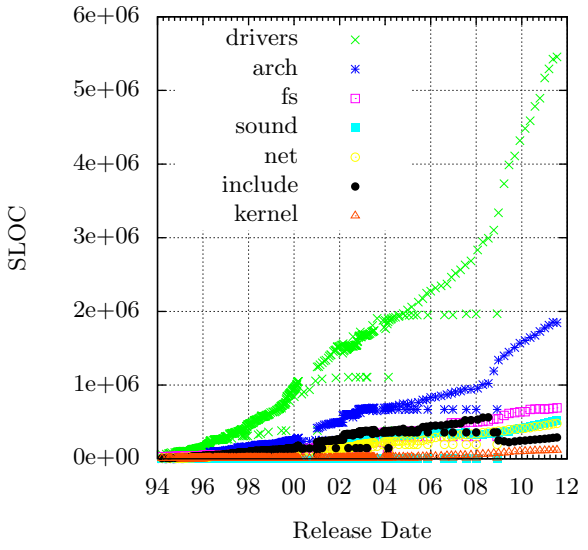

Figure 3: Major Sub-Directory SLOC for Linux Kernel 1.0 to 3.0

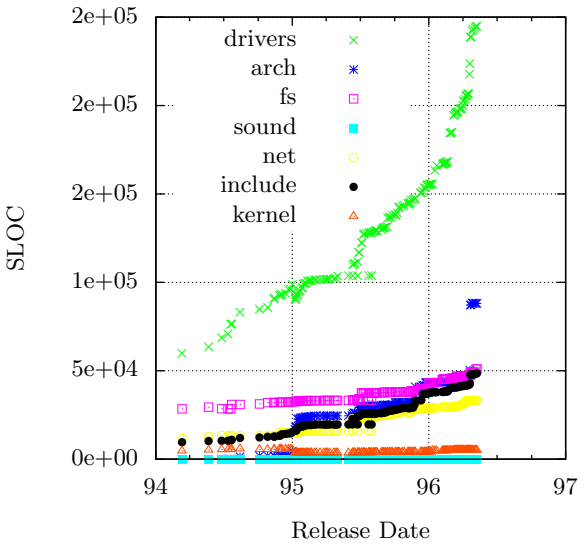

Figure 4: Major Sub-Directory SLOC for Linux Kernel 1.X

growing in bursts, however it may be more helpful to examine this more closely. The other sub-directories show growth, but are clearly must less significant in terms of SLOC than drivers and arch, however it can be seen that 'fs' is significant, and thus there must still be development in new filesystems such as ext4 and btrfs.

Figure 6.3 shows the same data as 6.3 but only showing data from kernel version 1.X. This is during more of the initial development, and we still see drivers as the largest part of growth in the kernel. The 'arch' directory is not present until around 1995, and not very significant until mid 1996. This may because initially only i386 was supported and it took some time before other platforms were supported and the code was added. The few other sub-directories (fs, sound, net) all grew at a steady rate as well.

Figure 6.3 shows similar data as figure 6.3 but only showing data from kernel version 2.X. Again, the 'drivers' directory is the significant factor in total SLOC. The 'arch' directory is growing steadily until around 2008 where it seems to increase in growth more quickly. Overall, we see 'drivers' and 'arch' being the major contributors to growth of SLOC.

6.4 Architectures

In order to more fully understand how the 'arch' sub-directory has grown with respect to release date, the sub-directories within 'arch' have been analyzed. Starting with version 1.0 which only supported i386 machines, to version 3.0 which supports 24 variants of processor architectures, it is clear there has been much development and growth in this area.

Figure 6.4 shows major architectures from version 1.0 to version 3.0 of the Linux kernel. Here we see x86/i386 as the major arch during the initial releases. Then powerpc increases along with arm in size. Around 2008 we see a

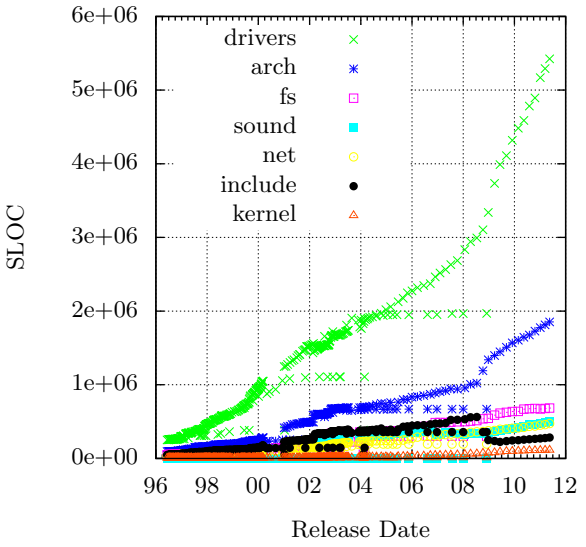

Figure 5: Major Sub-Directory SLOC for Linux Kernel 2.X

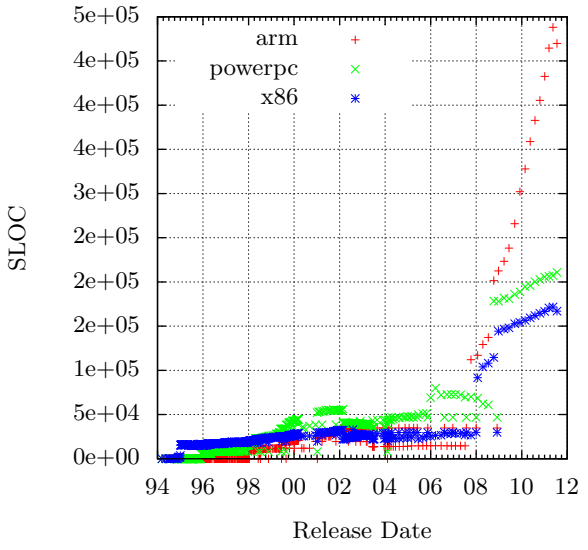

Figure 6: Major Architecture SLOC for Linux Kernel 1.0 to 3.0

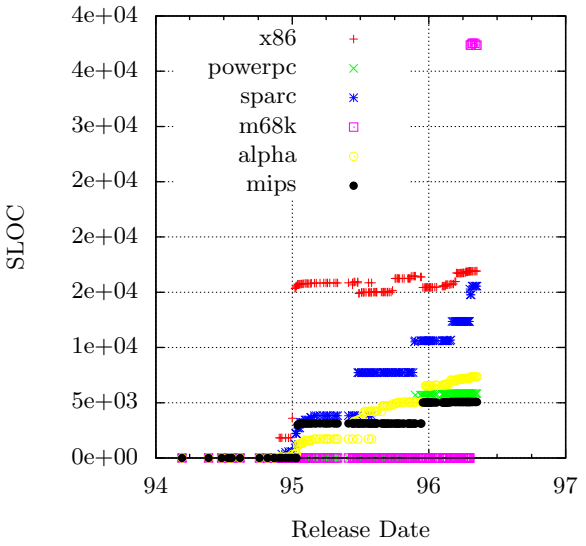

Figure 7: Major Architecture SLOC for Linux Kernel Versions 1.X

spike in all three architectures with arm growing at a very quick rate. At this point it would be helpful to zoom in on certain areas to get a better picture.

Figure 6.4 shows the major architectures for versions 1.X of the Linux kernel. Initially there are no 'arch' directories because the kernel only supports i386 architectures. The first ports of the Linux kernel were alpha, mips and sparc. This was soon followed by ppc/powerpc around 1996 and finally m68k. It is interesting to see some of the architectures progress in very fixed sized jumps. It would be interesting to find out if this is due to how the code was contributed to the kernel. For instance, if another company was responsible for developing this particular port, maybe their contributions coincide with the companies internal release schedule. The i386/x86 architecture seems to vary between each release which may be due to more regular contributions and changes. In addition, the alpha architecture seems to have been frequently updated.

Figure 6.4 shows the major architectures for just the stable Linux kernel versions (2.0, 2.2, 2.4). Initially, both development and stable versions were graphed on the same plot, but it was difficult to see the patterns so there were separated. Here we show an explosion of development in porting Linux to a number of platforms. As seen in Figure 6.4 it is also evident that arm has grown extremely rapidly from 2008 to the current version.

Figure 6.4 shows the major architectures for the developmental Linux kernel versions (2.1, 2.3, 2.5). Here we see a very volatile picture of SLOC changes and it seems appropriate that this is a developmental branch.

Figure 6.4 shows just the SLOC count for architectures in the 2.6 kernel version. Here we see pretty steady changes from 2003 to 2008. In 2008, arm and x86 development in-

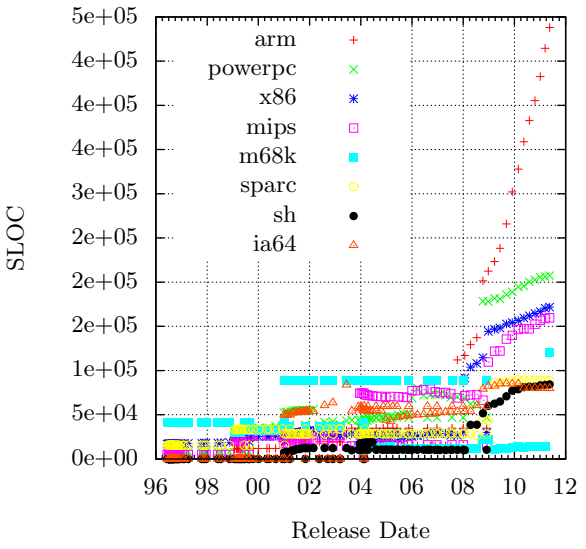

Figure 8: Major Architecture SLOC for Stable Linux Kernel Versions 2.0,2.2,2.4

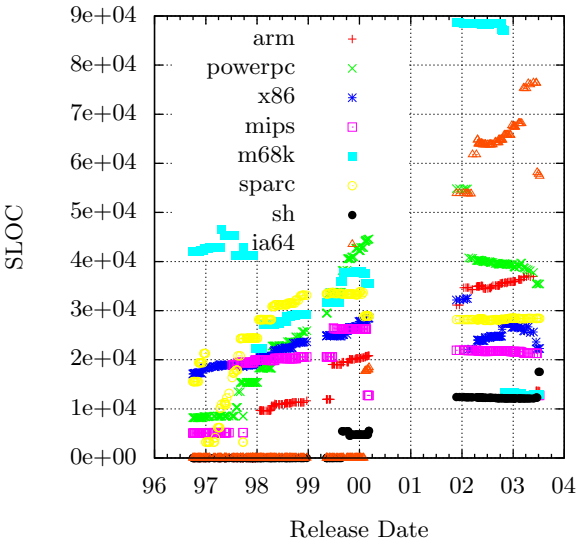

Figure 9: Major Architecture SLOC for Linux Kernel Versions 2.1,2.3.2,5

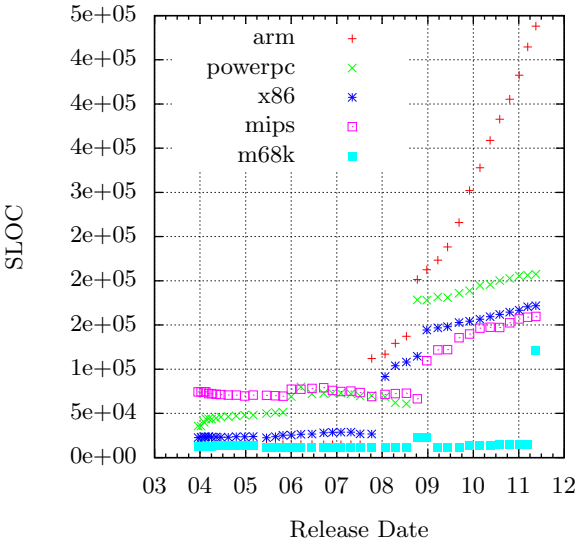

Figure 10: Major Architecture SLOC for Linux Kernel Versions 2.6

creases while a few others taper off. From 2009 until today we see a great change in arm development. In addition, powerpc, x86, and mips all seem to have increased quite a bit during this time period but not as great as arm. This growth makes sense because a great proportion of new phones that run Linux (Android) have Arm processors and thus need support for that particular architecture. However, this growth could pose a problem in maintainability and it seems that there is some awareness of this as read from this particular mailing list thread started by Linus Torvalds discussing the amount of code in the arm directory: `http://thread.gmane.org/gmane.linux.ports.arm.kernel/113895`.

7. CONCLUSIONS

Linux kernel versions from 1.0 to 2.0 were analyzed using SLOCcount to show how the source code has changed over all of these releases. In all 557 versions were analyzed and total as well as sub-directories were examined. Growth in sub-directories was explored as well as architecture support. Overall we can see where some major events and changes occur in the history of the kernel, and where potential issues could arise if growth seems to large. In particular its worth noting that the growth of the arm architecture seems to be much greater than the development of the rest of the kernel. In addition, the original developer of Linux, Linus Torvalds, has also expressed concern over this huge increase in SLOC in the arm architecture directory. This may be a good starting point to see where large increases in code over time may have caused issues, and to see how the community responds to such changes.

Overall, these results seem valid in that other studies show similar results. It is difficult to visualize all the data in one graph so many of these had to split into logical subdivisions. Overall the data collection took around 6 hours to download all the information, and around 12 hours to process on my Core 2 Duo system. The graphing and post-processing took a significantly smaller amount of time.

It is useful to use metrics like SLOC over release date, but must be understood in a larger context to be the most useful. Because versioning systems change, or major events in development may occur it is important to keep this in mind when drawing conclusions about any project.

7.1 Future Work

Since this is an exploratory study, there of course is much future work to be done.

First, it would be interesting to look at many other metrics than just SLOC. For instance, Cylomatic complexity, maintainability index, number of changes between versions, how many people contributed to a particular version, how many unique contributors to a section of code, and number of system calls. In addition to source code and revision control type of metrics, other types of metrics would be critical to gain a better understanding. Determining how many comments a particular patch had on a mailing list. It would also be good to utilize the kernel bugzilla or other Linux distribution bug tracking systems to see how many bugs were open and closed against various versions of the kernel. in addition to these types of metrics, it may be useful to gain some

insight into the actual social structure of the community. This could be showing how patches are contributed and information about maintainers. If patches are developed by a particular company, then this could be recorded. In addition recording the type of development practices of that company could also be relevant.

By using this host of information many things could be done. From a reliability and cost-saving perspective, this data could be mined and predictions could be made based on this data. For instance it may be useful to predict if a patch will have a higher probability of introducing bugs into the kernel, and if this is the case then it will need to be given extra scrutiny. In addition using all this data one could use data mining techniques to show which factors are more relevant in predicting failures and then do further studies concentrating on these factors.

7.2 Final Thoughts

Overall, this was a very interesting study and much insight was gained in how to do a case study. While picking a topic was not as difficult, gathering and analyzing data could be rather time consuming. This study shows that much more data could be explored given ample time.

8. REFERENCES

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

[2] Sloccount. `http://www.dwheeler.com/sloccount/`.

[3] A. Israeli and D. Feitelson. The linux kernel as a case study in software evolution. *Journal of Systems and Software*, 83(3):485–501, 2010.

[4] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski. Metrics and laws of software evolution-the nineties view. In *metrics*, page 20. Published by the IEEE Computer Society, 1997.

[5] G. Robles, J. Amor, J. Gonzalez-Barahona, and I. Herraiz. Evolution and growth in large libre software projects. In *Principles of Software Evolution, Eighth International Workshop on*, pages 165–174. IEEE, 2005.

[6] D. Wheeler. Estimating linux's size. *http://www.dwheeler.com/sloc*, 2001.

APPENDIX

A. SLOC DOWNLOAD AND PROCESS SCRIPT

```
#!/bin/bash

SERVER_PATH="http://mirror.anl.gov/pub/linux/kernel"
SLOC_CMD="sloccount"
SERIES="1.0 1.1 1.2 1.3 2.0 2.1 2.2 2.3 2.4 2.5 2.6 3.0"

download() {
        for series in $SERIES; do
                curl -# -f --digest --compressed -O \
                -R ${SERVER_PATH}/v${series}/\
                        linux-${series}.tar.gz
                filename="linux-${series}.[0-132].tar.gz"
                curl -# -f --digest --compressed -O \
                -R ${SERVER_PATH}/v${series}/${filename}
        done
}

sloc() {
        for file in $(ls *.tar.gz); do
                name=${file%.tar.gz}
                echo Analyzing $name ...

                # Write date to top of file
                stat -f "%m%t%Sm %N" ${file} \
                        | cut -f2- > ${name}.sloc

                # Extract archive
                tar -xf $file

                # Determine directory name
                if [ -d "./linux" ]; then
                        dir="./linux"
                else
                        dir=${name}
                fi

                # Run code metrics
                $SLOC_CMD ./$dir  >> ${file%.tar.gz}.sloc

                # Run addition code metrics
                $SLOC_CMD ./$dir/arch  \
                        >> ${file%.tar.gz}.arch.sloc

                # Cleanup directory
                rm -rf $dir
        done
}

case "${1}" in
        download)
                download
                ;;
        sloc)
                sloc
                ;;
        *)
                echo "Usage: $0 [download, sloc]"
                exit 1
                ;;
esac
```

B. SLOC PARSE SCRIPT

```
#!/usr/bin/perl

@subdirs=('drivers', 'arch', 'fs', 'sound', 'net',
'include', 'kernel', 'mm', 'tools', 'crypto',
'security', 'scripts', 'lib', 'block', 'Documentation',
'ipc', 'virt', 'init', 'firmware', 'samples',
'arm', '(powerpc|ppc)', '(x86|i386)',
'(mips|mips64)', 'm68k', '(sparc|sparc64)',
'sh', 'ia64', 'blackfin', 'cris','s390', 'alpha',
'parisc', 'tile', 'um','mn10300', 'frv','avr32',
'm32r', 'xtensa', 'microblaze', 'unicore32', 'h8300',
);

# Print header
print "# ker maj min date total @subdirs\n";

# Read each .sloc file
@files = <*.sloc>;
foreach $file (@files) {
        # Extract version information
        $name = $file;
        $name =~ s/\.sloc//;
        $name =~ s/linux\-//;
        print "$name ";

        # Open each file
        open SLOC, "<", $file;
        @file = <SLOC>;

        # Get Date      Jan  5 07:54:14 2004
        ($M, $d, $t, $Y, $x) = split(' ',@file[0]);
        print "$d-$M-$Y ";

        # Extract total number of lines
        foreach $line (@file) {
                chomp($line);
                if ($line =~ /Total Physical Source Lines of Code \(SLOC\).*/) {
                        ($x, $tot) = split('=',$line);
                        $tot =~ s/([, ])+//g;
                        last;
                }
        }
        print "$tot ";

        # Extract other information
        foreach $sub (@subdirs) {
                $subtot=0;
                foreach $line (@file) {
                        if ($line =~ /[0-9]+ +$sub/) {
                                ($subtot, $x) = split(' ', $line);
                        }
                }
                print "$subtot ";
        }

        print "\n";

        close(FILE);
}
```