Evolution and Growth in Large Libre Software Projects*

Gregorio Robles, Juan Jose Amor, Jesus M. Gonzalez-Barahona, Israel Herraiz
GSyC, Universidad Rey Juan Carlos (Madrid, Spain)
{grex,jjamor,jgb,herraiz}@gsyc.escet.urjc.es

Abstract

Software evolution research has recently focused on new development paradigms, studying whether laws *found in more classic development environments also apply. Previous works have pointed out that at least some laws seem not to be valid for these new environments and even Lehman has labeled those (up to the moment few) cases as anomalies and has suggested that further research is needed to clarify this issue. In this line, we consider in this paper a large set of libre (free, open source) software systems featuring a large community of users and developers. In particular, we analyze a number of projects found in literature up to now, including the Linux kernel. For comparison, we include other libre software kernels from the BSD family, and for completeness we consider a wider range of libre software applications. In the case of Linux and the other operating system kernels we have studied growth patterns also at the subsystem level. We have observed in the studied sample that super-linearity occurs only exceptionally, that many of the systems follow a linear growth pattern and that smooth growth is not that common. These results differ from the ones found generally in classical software evolution studies. Other behaviors and patterns give also a hint that development in the libre software world could follow different laws than those known, at least in some cases.*

1. Introduction and research goals

The number of studies on software evolution is relatively low, despite being a field opened more than 30 years ago. The lessons learned are many, and are summarized in a set of laws, stated by Lehman, which have grown to eight in their latest version [17]. These *laws* have been validated empirically with some large industrial software projects. Recent research is exploring whether they are applicable to other domains, such as systems developed using eXtreme Programming models, based on the COTS paradigm, etc.

One of this 'other' domains is libre software[1]. Although its basic difference with 'traditional' software lies in the licensing terms, many argue that there are also significant differences in the way they are built. For instance, most of the procedures in libre software are open and public, targeted to ease the followup and joining by new developers, with the aim of forming a developer community in which individuals can play several roles (from core developers to casual bug report submitters). Although there is some literature showing that projects with a surrounding community are exceptions if we consider the whole libre software landscape [13], they are still the most notorious, larger in size and user population, and those which have featured most attention by the public, the industry and the research community (consider for instance Mozilla [6, 18, 19], Linux [9, 19, 21], Apache [18], GNOME [14, 8], or FreeBSD [4]).

For the study presented in this paper, we have considered exactly this kind of libre software projects: large in size (at least in the order of 100K lines of code), and with a large user and developer community. Our intention is to explore how they behave in the context of the laws of software evolution, specially regarding software growth. For this matter, we started by reproducing (with current data) the classical study performed five years ago on the Linux kernel [9], which seemed to question the conformance of libre software projects to some of those laws. Later, we extended the study by doing a similar analysis on other libre software systems in the same domain (operating system kernels): the *BSD family. Fi-

*This work has been funded in part by the European Commission, under the CALIBRE CA, IST program, contract number 004337, in part by the Universidad Rey Juan Carlos under project PPR-2004-42 and in part by the Spanish CICyT under project TIN2004-07296.

[1] Through this paper we will use the term "libre software" to refer to any code that conforms either to the definition of "free software" (according to the Free Software Foundation) or "open source software" (according to the Open Source Initiative).

nally, we performed again the same kind of study for 18 other large libre software applications, in order to find out whether the results found are general or not for this domain.

In all these cases, we also had another goal in mind: to look for differences in the evolution of the software before and after the 1.0 release. Traditional software evolution studies consider only releases after the first one delivered to the customers (usually 1.0). However, it is a common behavior in many libre software project to follow the rule "release early, release often", which means that programs are available to the public well before they are considered stable, and that the first release named "stable" is not that special. Therefore, it is difficult to find a point where "evolution" starts, so we studied (if available) the whole life cycle and hoped to find the significance of the 1.0 release from our results.

The rest of this paper is structured as follows. The next section references the previous work on software evolution, including studies related to libre software. The third section details the methodology used in the presented study. After that, the main results of applying it to the Linux kernel and its subsystems are shown. In the fifth section, the same is done for the family of *BSD kernels, while the sixth one is devoted to discuss whether the findings in those systems can be generalized to other large libre software cases. In the final section, some conclusions are drawn.

2. Previous research

Thirty years of research on software evolution have resulted in a set of *laws*, known as Lehman's Laws of Software Evolution [15]. Although the number of laws has grown from three in the seventies to eight in their latest version [17], all of them have been empirically proved, by studying projects developed in traditional industrial software development environments.

One of the laws that is more related to the study we are presenting is the Fourth Law [17], which states that the rate of development over the life of a program is approximately constant, and independent of the resources devoted to it. Both Lehman and a statistical study performed by Turski [23] found that those software systems follow an inverse square growth rate. The equation given by Turski is:

$S_i = S_{i-1} + E/(S_{i-1})^2$

where S_i is the estimated size of the system at the i-th release (in number of source modules[2]) and E is a parameter. An explanation that is given for this equation states that for a system of size *n* (modules), the maximal number of possible interconnections is $n \cdot (n-1)$. As the system grows, introducing new modules will impact a growing number of existing ones and more effort will be required [16].

When solved directly, the equation is approximately

$S = (3E \cdot t)^{1/3}$

where S is the size of the system measured in modules, t is time and E a parameter.

In the specific field of libre software, there are some research works from the point of view of evolution: Burd et al. [1] evaluate the evolution of GCC, a compiler collection written mainly in C; Capiluppi, alone or with colleagues, has also authored several works about the evolution of libre software projects [3] and has proposed some models [2] (although those studies are focused on small to middle-sized projects). However, the most relevant study on the evolution of libre software projects is probably the one by Godfrey and Tu [9], who studied the Linux kernel in 2000. They found that Linux, then about 2 million lines of code in size, had a super-linear growth rate, apparently in contradiction with Lehman's Fourth Law, and with the statistical evidence from Turski.

The main conclusions of this work can be summarized as follows:

- The Linux kernel exhibits a super-linear growth rate. Most of the growth is due to new functionality and added hardware support, not to bug fixing.
- Much of the functionality (specially device drivers) is complex and extensive, but also relatively independent from each other, and from the rest of the system.
- External contributions (both for adding and maintaining code) were frequent in the devices and architecture subsystems. Maintenance is often done by third parties.
- Large parts of the kernel (specially device drivers) do not require active maintenance, but are still shipped with Linux just in case the user needs them.
- Fourth Lehman's law of software evolution is presumably not fulfilled in the case of Linux.

In a later paper, the same authors propose the following software growth equation, based on statistical analysis [10]:

$y = 0.21 \cdot t^2 + 252 \cdot t + 90,055$

[2] Although there is no precise definition of what a module is as it varies from system to system, in general a module refers to an individual source code file.

where y is the size in uncommented lines of code and t the days since version 1.0. The coefficient of determination, calculated using least squares, has a value of r^2 = .997.

There is also a study by Succi et al. [22] about the growth in libre software systems, which confirms this super-linearity for the Linux kernel, but finds linear growth for GCC and Apache.

Both studies have been considered by Lehman et al. [16] as anomalies of the 'laws' of software evolution affecting libre software, at the same time that they encourage further research on this topic. This paper follows that advice, providing further empirical insight in that direction.

3. Methodology

The methodology used in this work is based on analyzing source code publicly available on the Internet. The code corresponding to every snapshot considered is downloaded to a local directory, where its size is computed. The results are stored in a database, which is later used for plotting and performing a detailed analysis.

There are some differences with respect to which kind of public repositories are used for obtaining the source code, and which snapshots are considered, depending on what is available for each project. Details will be sketched below for the projects considered.

The size of each snapshot is obtained by using SLOCCount[3], a tool which uses some heuristics to identify files with source code (and the language in which they are written), and to compute the number of physical source lines of code (SLOC) they contain. For this purpose, physical SLOC are defined as "a line that finishes in a mark of new line or a mark of end of file, and that contains at least a character that is not a blank space nor comment". SLOCCount is a mature tool that has been used also for analyzing other systems, such as complete GNU/Linux distributions with several dozens of millions of source lines of code [11, 12].

In the case of the Linux kernel, there is no public CVS repository available. Therefore we decided to download release packages. All Linux releases are available in Linux mirrors, where they are packed as compressed tar files. We got the official and experimental kernel releases, from 1.0 to the last one published in December 2004 (2.6.10). We have also measured those known as "historic" releases, that is, those released prior to 1.0, although they are considered to be unstable, and suffered from frequent reorganizations in the code base.

In order to recreate Godfrey's study on the Linux subsystems, we did not only compute the number of source lines of code for the whole system, but we also gathered data from all main subdirectories (which we will call subsystems from now on).

For all other systems considered in this paper, public CVS repositories are available. In libre software projects, it is common practice that even being unstable, the software in the repository can be compiled and is usable, up to the point that automatically generated nightly-builds are offered in many cases, i.e. it is not in a state of flux. Releasing a new version of the software consists usually in taking one of this snapshots and assigning it a specific release name/number, although some projects have more sophisticated procedures [5].

Taking these facts into account, we retrieve monthly snapshots from the CVS repositories, starting by the time the repository was established, until April 2005. The whole process has been automated and can be obtained as a publicly available tool [20]. For *BSD kernels, only kernels (directory src/sys) are considered, for the rest, the whole CVS module is studied.

Although Lehman suggests plotting software size against release numbers, we have done it against time, for two reasons. First, we feel this way matches better the semi-continuous release process found in many libre software projects. And second, this way of depicting evolution was also the one used by Godfrey et al., for Linux. This also implies that using periodic CVS snapshots is enough, and we do not need the source packages for specific releases.

Another sensible difference with Lehman's studies lies in the metric used for software size. Lehman uses modules, because he argues this metric is more consistent (it has a "higher degree of *semantic integrity*") than considering source lines of code [16]. Godfrey and Tu counted uncommented lines of code, and we will do the same. However, preliminary studies on the Linux kernel seem to imply that the mean size of modules (counted in lines of code) remains almost constant in time; we have observed the same behavior in a fast inspection for some projects. This would imply that counting lines or modules would give the same evolution patterns. However, further research is needed to verify if this is valid in general and to study the correlation of both metrics if not.

As a final note, it is worth mentioning that in some cases, the projects under study started outside a CVS, but were later uploaded to one. In those cases, an initial

[3] SLOCCount has been developed by David A. Wheeler, and is libre software. It is available from `http://www.dwheeler.com/sloccount/`.

gap will appear in the plots.

4. Observations on the Linux kernel

Table 1 shows the main differences between the analysis on Linux by Godfrey and Tu, and the one we have performed. When the first study was performed (in the year 2000), two concurrent Linux versions existed: the stable version 2.2.14 and the development version 2.3.39. 34 of the 67 stable releases, and 62 of the 369 development releases were analyzed in it, totaling 96 releases. In our study, we have considered all the releases published, both stable and development, including those prior to the considered first stable release (1.0), and up to 2.6.10 (released December 24th 2004). All in all, we have studied 123 stable and 457 development releases. It should be noted that even if the 2.6 branch is the bleeding-edge stable branch, previous stable branches (2.0, 2.2 and 2.4) are still actively maintained (although usually without addition of new functionality, only bugs are removed) and new releases appear from time to time. The number of lines of code in 2.6.10 is larger than 4 millions, with a tarball size of about 45.5 MB. These figures can be compared to those of 2.3.39: about 1.5 million lines of code, and about 17 MB of tarball.

	Godfrey & Tu	Our study
Date	Jan 2000	Dec 2004
# of releases	369D + 67S	457D + 123S
Studied releases	62D + 34S	457D + 123S
Most recent ver.	2.2.14 (2.3.39)	2.6.10
KSLOC most recent	1,425 (1,607)	4,147
Tarball size (MB)	15.9 (17.7)	45.5
Counting	wc + awk	SLOCCount

Table 1. Comparison between Godfrey & Tu's [9] and our study.

In the next subsections we will show the results of reproducing Godfrey and Tu's study for the growth of Linux system-wide and for its subsystems (with the already mentioned slight methodological differences).

4.1. System level growth for Linux

Figure 1 shows the growth of Linux in terms of sources lines of code, from the first versions in 1991 to the most recent one in 2005. We have depicted two vertical lines in all figures for Linux, showing the release date of the 1.0 version (in the year 1994) and the time of the study by Godfrey and Tu (in the year 2000). It can be noted that the super-linearity that was found by Godfrey and Tu seems at first glance to have become more remarkable with time. Based on statistical analysis we have obtained the following software growth equation:

$y = 0.26 \cdot t^2 - 322 \cdot t + 195,183$

where y is the size in source lines of code and t the days since version 1.0. The coefficient of determination computed using least squares is r^2 = .990.

Compared to Godfrey and Tu's equation from 2000, we see that both are quite similar. Interestingly enough, our initial inspection, which led us assume that the super-linear growth has become more remarkable with time, is demonstrated by the fact that the factor that multiplies the quadratical growth is now 0.26 instead of 0.21, meaning that the growth of Linux has accelerated during the last five years.

Another fact that can be observed from this figure is that the growth pattern followed by the Linux developers has changed over time. Until the year 2000 we find a strong growth in the development branches, with stable branches coming out of it with almost horizontal evolutions (showing almost stagnation in growth), while the newer stable branches 2.4 and 2.6 show steep growths. That way, the current stable branch (2.6) is growing steadily, with a shape quite similar to a development branch. However, the latest releases seem to slow down, maybe foreseeing the stabilization that would occur just before the start of a new development branch (which would be 2.7).

Graphs for the evolution in time of the tarball sizes or the number of files have been omitted for lack of space, but they show the same behavior in terms of growth as the number of lines of code shown in figure 1.

4.2. Growth of major subsystems

Godfrey and Tu included in their study the analysis of the major Linux subsystems, following an idea by Gall et al. [7], who stated that looking at the evolution of subsystems could bring more insight about the software under consideration.

The growth of major subsystems can be seen in figure 2. As in the original work, the most growing subsystem is the one comprising drivers, which grows steadily even though in version 2.5.x the *sound* subsystem was taken apart (which justifies the ripple around 2002).

Godfrey and Tu plotted the evolution of the share of code by the major subsystems from version 1.0 (in 1994), while we also studied the previous versions although we have not included that figure in this paper

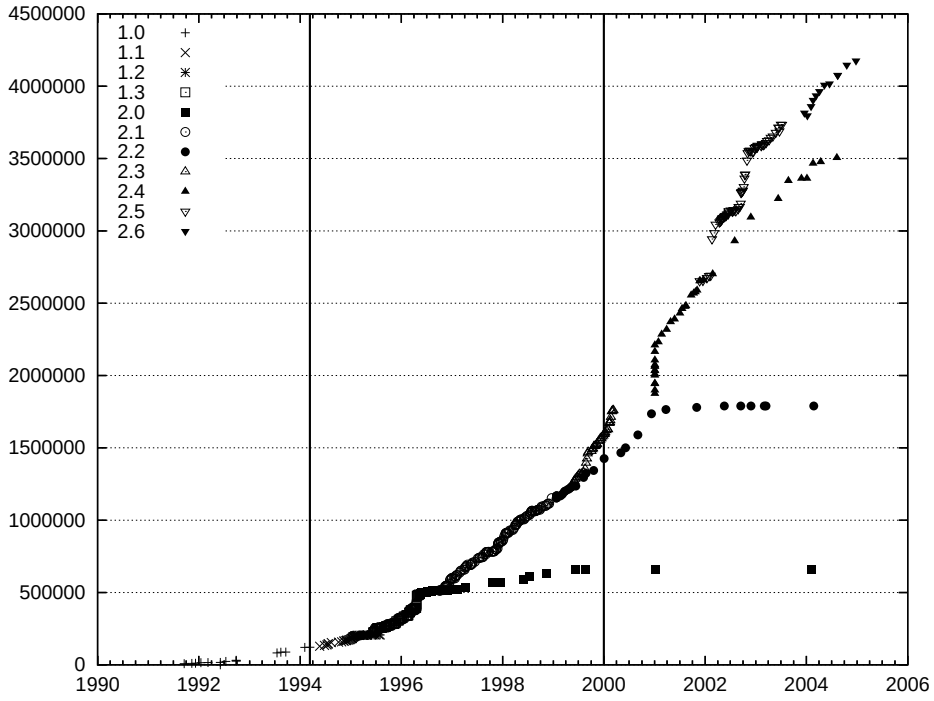

Figure 1. Growth (lines of code) of Linux

for the sake of space. Those early releases show an erratic behavior, because the architecture for Linux was at that time not specified and changed drastically several times. But from version 1.0 onwards, all major subsystems show an almost parallel growth pattern, meaning that their relative growth is similar. Again, the gap that can be found in early 2002 is due to the removal of *sound* from *drivers*, while the one in early 2001 is due to some code being allocated to the *arch* subsystem, as it can be clearly observed from the figure. Besides these inconsistencies, we can observe how the share of code corresponding to the *drivers* subsystem has remained almost constant since 2000, while in the period from 1998 to 2000 its share grew from around 50% to 60% of the total kernel size, even when Linux itself doubled its size. On the other hand, *net* and *fs* show a decreasing share, although it seems that in the latter case its presence has remained around 10% during the last seven years, while *include* and *sound* remain almost constant in time through the whole system life, since version 1.0.

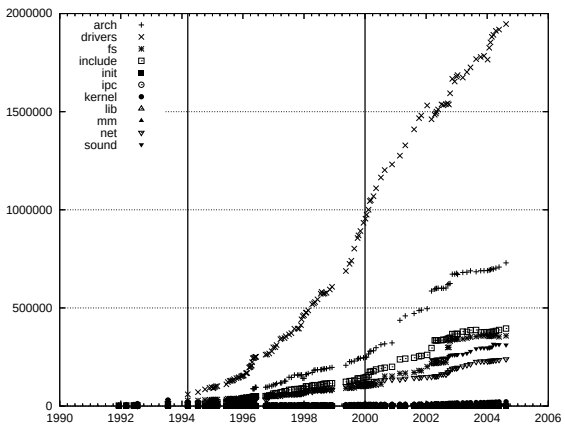

Figure 2. Growth of the major subsystems in Linux (development releases)

If we filter out the *drivers* subsystem from figure 2, we can identify the *arch*, *fs*, *include* and *net* subsystems and see how their growth shows a super-linear pattern. This occurs even in the case of *net*, which has a growth

that is not that steep as the one of the others. Hence, we can conclude super-linear growth patterns can also be found at the subsystem level in Linux.

Figure 3 shows the growth of the smaller, but most essential subsystems of Linux: *init*, *ipc*, *kernel*, *lib* and *mm*. The behavior before 1994 (version 1.0) is again chaotic for the *kernel* subsystem. Much of its code was later moved to other subsystems, although since 1995 it shows a super-linear growth almost equal to the one exhibited by the *mm* subsystem. The rest of the subsystems do not show clear growth patterns: besides *lib*, which has recently started to show a clear tendency to grow, the rest remain almost constant for a long time, although and from time to time they are affected by small gaps because of code inclusion or exclusion. In any case, all these subsystems are relatively small, which makes their growth patterns less significant than those considered before for the system-wide analysis.

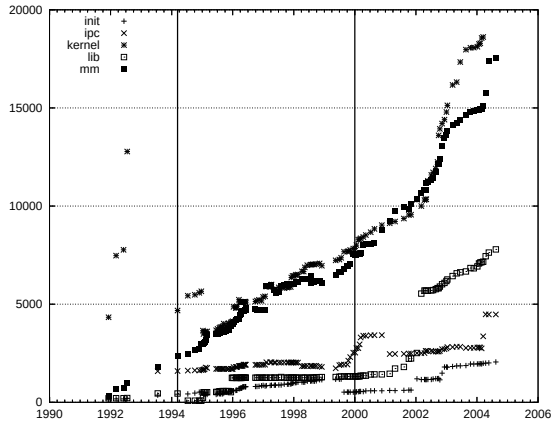

Figure 3. Growth of the smaller, core subsystems in Linux (development releases)

Performing our study with smaller granularity, we can study the subsystems that compose the *driver* subsystem, which is by far the most important in terms of lines of code. Interestingly enough, for almost all those subsystems we couldn't see hardly no super-linear growth. The only curve that shows such a behavior is the one that groups the "rest" of the drivers (those that are not one of the major ones). The rest show linear trends with periods of high activity, when a lot of code is included at once (as for instance in early 2004 for *scsi*), or regularly (the *net* subsystem shows a pronounced growth from mid 1999 to early 2000, clearly different from the linear trend before and after that period). It can be observed that the sum of *other* surpasses it. But none of the subsystems in *other* surpasses *net*; what happens is that their number has raised to 37 more than those studied by Godfrey and Tu.

In general, and in all subsystems, when we perform a study on a smaller granularity level, super-linearity gets less and less frequent and linearity arises. Godfrey and Tu pointed out the existence of independent development groups that worked in parallel due to a high modularization of the Linux kernel. We are currently actively researching if the linear growth patterns that arise at a detailed level of analysis correspond to these development groups. This could mean that each subsystem would behave as a whole, independent system, with its own (linear) growth pattern. In that case, the whole kernel is just the composition of independent behaviors throwing a super-linear pattern as a raise in the number of subsystems can be observed.

5. Observations on the *BSD kernels

The operating systems based on the BSD kernel constitute the most similar alternative to Linux-based operating systems in the libre software world. All BSDs derive from the UNIX version made at Berkeley since the 1970s. In particular, both FreeBSD and NetBSD are derivatives of the 4.4 BSDLite version released 1994, while OpenBSD is a branch (first released 1996) from NetBSD. These three BSD derivatives share architecture and a lot of code [24], so 'copying' source code, or even entire modules from the other kernels is common practice.

As in the case of the Linux kernel, we have researched the growth of each of the BSD kernels as a whole, and at the subsystem level.

5.1. System level growth for the *BSD kernels

Figure 4 gives the system growth for these systems starting in late 1995, when OpenBSD had its first commits. Even though all three kernels have achieved a significant size (2.5 MSLOC for NetBSD and over 1.5 MSLOC for OpenBSD and FreeBSD), we can see that their growth is not super-linear.

NetBSD and FreeBSD show an almost linear growth pattern (see the values of the determination coefficient r^2 in table 2), which OpenBSD follows too, but only until 2001 (afterwards it loses large quantities of code in two occasions). Interestingly enough, we can identify a super-linear growth rate for FreeBSD until the year 2000, which means that if Godfrey and Tu would have

BSD kernel	Growth equation (linear fit)	r^2
NetBSD	$y = 610.2 \cdot t + 585731$	.993
FreeBSD	$y = 479.1 \cdot t + 2000607$	.972
OpenBSD	$y = 240.2 \cdot t + 779607$	.891

Table 2. Growth equation for the BSD kernels (based on statistical analysis)

performed their study also on FreeBSD, they would have found at the time of writing their paper similar patterns for both of them. Only Linux keeps with such a super-linear growth, while FreeBSD seems to follow a more linear shape.

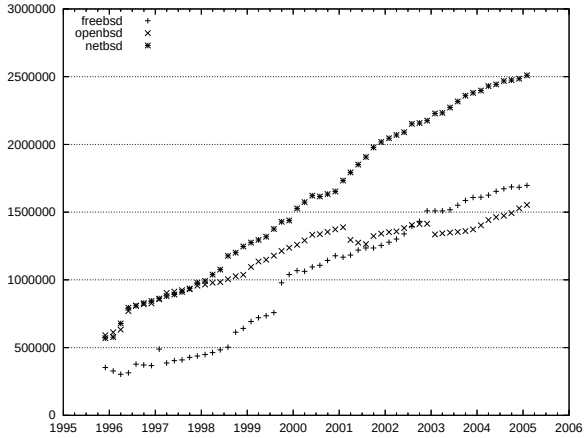

Figure 4. Growth of the BSD derivatives

5.2. Growth at the subsystem level

Figure 5 shows the growth of the subsystems for FreeBSD kernel in the same way we have done in the previous section for the Linux kernel. The equivalent figures for the OpenBSD and NetBSD kernels have been omitted for the sake of space. Subsystems do not grow super-linearly in any of the three cases, except for *dev*, and only in the early times. It is noteworthy that the shape of the *dev* subsystem is similar in all cases, possibly due to a common code base.

The *arch* subsystem is the largest one both in NetBSD and OpenBSD, although in the latter case it seems to stop growing early in 2001 contributing a great deal to the shape to the total OpenBSD system.

Since one of the main goals of NetBSD is to work on as many platforms as possible, the larger size of *arch* isn't a surprise, neither its continuous growth. OpenBSD supports many architectures, but with less drivers, and is the less growing system of the three, probably due to its strict security/auditing policies (as it is the base of an operating system targeted to environments with strict security constraints).

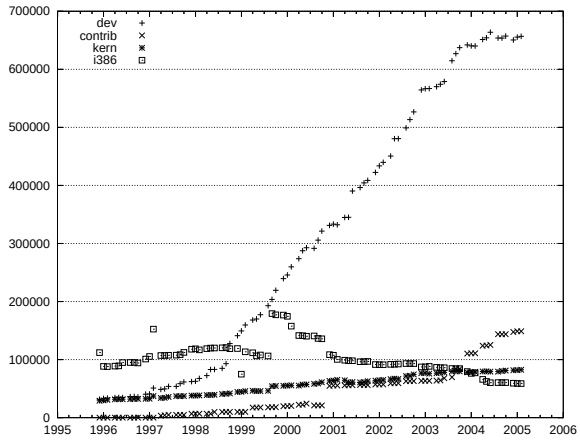

Figure 5. Growth of the four largest subsystems of FreeBSD

6. Observations on other systems

We have studied 18 more large libre software systems, to widen the sample, with the aim of applying the methodology to more cases, and to find out if results can be generalized. We have focused on projects which can be considered mature, and with an active community of users and developers. In particular, we have selected projects with a more an ample set of contributors (in the range of the hundreds or above), since a critical mass has to be achieved to ensure sustainability of large projects (even if the contributions are unequally distributed) [18].

All of the selected systems are related to well known libre software projects: GNOME, KDE (both aimed at building a complete desktop environment), Apache (well known by its web server, but producing also many other tools) and Mono (an implementation of .NET). These projects are usually considered typical libre software ones, although several different development models are found among them. However, all of them include voluntary and paid development work, external contributions, and interest in satisfying the needs of a sizable user community.

Project	Start	Version 1.0	Prev.	Ripples	Size	Growth Function	Corr. coef.
kdelibs	May 97	Jul 98	N	Y	615K	$y = 6421.1 \cdot t + -16474.8$	r = 0.995
jakarta-commons	Mar 01	-	N	N	429K	$y = 9394.7 \cdot t + -33888.0$	r = 0.994
mcs	Jun 01	Jun 04	N	N	1081K	$y = 26002.3 \cdot t + -105089.3$	r = 0.993
mono	Jun 01	Jun 04	N	N	222K	$y = 4912.9 \cdot t + -3436.6$	r = 0.992
koffice	Apr 98	Jan 01	N	N	780K	$y = 7965.3 \cdot t + 20724.8$	r = 0.992
kdepim	Jun 97	Jul 98	N	N	512K	$y = 4920.4 \cdot t + -32103.6$	r = 0.990
gnumeric	Jul 98	Jun 02	N	N	229K	$y = 3019.9 \cdot t + 17322.8$	r = 0.988
gtk+	Dec 97	Apr 98	Y	Y	388K	$y = 3371.7 \cdot t + 89968.9$	r = 0.985
xml-xerces	Nov 99	Oct 03	Y	Y	375K	$y = 4345.2 \cdot t + 104761.5$	r = 0.977
galeon	Jun 00	Dec 01	N	Y	90K	$y = 1460.0 \cdot t + 9095.4$	r = 0.967
httpd-2.0	Sep 99	Sep 02	Y	Y	127K	$y = 1000.5 \cdot t + 65668.8$	r = 0.947
xml-xalan	Nov 99	Oct 00	N	Y	337K	$y = 3896.0 \cdot t + 101817.1$	r = 0.943
kdebase	Apr 97	Feb 99	N	Y	362K	$y = 3097.1 \cdot t + 72062.8$	r = 0.935
kdenetwork	Jun 97	Jul 98	N	Y	293K	$y = 2142.9 \cdot t + 48781.0$	r = 0.933
kdevelop	Dec 98	Dec 99	N	Y	386K	$y = 4146.8 \cdot t + -22622.4$	r = 0.916
ant	Feb 00	(Aug 03)	N	Y	120K	$y = 1774.4 \cdot t + 15212.3$	r = 0.882
evolution	May 98	Dec 01	N	Y	208K	$y = 3801.2 \cdot t + 35801.7$	r = 0.842
gimp	Dec 97	Jun 98	Y	Y	557K	$y = 2696.7 \cdot t + 317718.9$	r = 0.815

Table 3. Summary of the evolution of the libre software systems under study.

The data for these systems has been obtained in April 2005, and at least four years of development are considered for all of them. Table 3 includes a summary of the evolution of the 18 libre software systems studied. For each row, the data about one system (its name can be found in the first column) is offered: the date when the system started to use CVS (not the starting date of the project); the date for the 1.0 version, if available (in some cases that data is not available, such as jakarta-commons, because it groups a set of subcomponents that are released independently or in the case of Ant for which we haven't found a 1.0 release, so we have inserted the date for release 1.1); whether a code base existed before starting the system's CVS (such as for GTK+ or The GIMP); whether strong ripples can be observed in the growth of the system; the current size of the system (in lines of code); the linear growth function that fits the data for the system where t is given in months and y in lines of code; and finally the last column contains the correlation coefficient for that fitting.

From the 18 projects that have been considered, at least 9 of them show a clear linear behavior throughout all the systems life. We have fitted them to a linear function with r (correlation coefficient) values of about .99 and .98. However, all these systems have quite different sizes and start dates, which reflect in different slopes for their growth lines. It is also interesting to notice that for these projects it is not possible to infer from its growth plot when version 1.0 was released as the pattern is the same before and after that release.

From the remaining 9 projects, 6 of them show also a linear trend, although (strong or frequent) ripples in their growth curve cause them to be fitted to linear functions with r below .97 (but in all cases above .91). If those ripples are filtered out, many of them show a behavior which is similar to the one observed for the first group. Ripples are usually due to the inclusion of external code, the restructuring of the code base or the removal of code.

The remaining three projects, Ant, GIMP and Evolution, show growth patterns that clearly cannot be fitted to linear (in fact, they show *bad* r values of .88, .87 and .80). While Ant shows a classical smooth growth (with some ripples early in the year 2002) as found in traditional software evolution studies up to the moment, the other two projects may be seen as exceptions or anomalies and hence require further explanation. Evolution started as a small community-driven project, but about two years later it was identified by a software company as a key software for its business model and devoted several developers to it. This may explain the super-linear growth trend in its first stages, until version 1.0 was released. After that point, the growth follows the usual pattern identified by Turski, except for the heavy refactoring that has made the code base get smaller at least two times in the latest stages of development. On the other hand, GIMP was uploaded to the CVS only after three years of development, already with about 300,000 lines of code, which may cause some distortion. In any

case, until about 2000 we can observe an almost linear pattern. But since then its growth has stagnated, in part because it has a mature architecture, and most of the development around GIMP is happening in modules, outside what is considered GIMP itself.

Therefore, we can say that 16 out of 18 systems follow a growth pattern linear or close to linear, and those that do not can be considered special in some sense.

To finish this section, just some considerations about the ripples found in the second group. The reader should remember that the systems we are considering are a part of larger projects, which means that code restructuring may happen not only intra-project, but also inter-project. For instance, a sudden gap downwards may mean that some large part of the source code has been pulled out from a project to start a more specific one. This behavior has been previously reported in the research literature on libre software: so, for instance, for the sake of modularity if the core group of developers grows larger than 15 to 20 developers, the project is split into smaller projects with the intention of improving manageability [18].

7. Conclusions

In this paper we have shown how Linux continues exhibiting a global super-linear growth pattern, as was noticed by Godfrey and Tu five years ago. However, super-linearity has become even more clear during those five years. At the subsystem level, the *drivers* subsystem shows to be the most important component, being itself the aggregate of many different smaller components (device drivers), usually built by different groups of developers. Most of those device drivers show a linear growth, but the number of device drivers is increasing, leading to a total super-linear growth pattern.

Applying the same study to the kernels of the BSD family, we have found that they are generally not growing super-linearly. However, before the year 2000 the growth of FreeBSD was super-linear, and the same can be said for some of its subsystems, and those of NetBSD. Except for these cases, the most predominant software growth pattern is the one that follows linearity.

The appearance of super-linear patterns seem to be related to the sudden inclusion of external code, to the existence of residual old code that does not need to be maintained (for instance, drivers for old devices), to a specific software architecture design (with an already fixed specification which has been widely tested, and therefore only coding has to be done), or to the allocation of work to different development teams. In this sense, further research should be performed on the linearly growing subsystems and find out if super-linearity is only achievable by aggregating work from different parallel-working development groups. In any case, it is not surprising that the Linux kernel shows such a pronounced super-linear growth as it features most (if not all) of the above characteristics.

With the aim of finding whether these results can be extrapolated to other (non-kernel) domains, we have also studied the growth of 18 large libre software applications. We have found that most of them show a clear linear growth pattern, in some cases after filtering out some ripples, due to occasional addition or removal of large quantities of code.

Therefore we can conclude that, for the sample analyzed, growth is usually linear, with some cases of super-linear growth. Since the sample is reasonably large, and representative of a certain kind of libre software systems (large in lines of code, with an active community and user base, stable) we believe that this can be considered the common growth pattern for this kind of systems.

We can also conclude that there are no noticeable differences in the growth pattern before and after the first stable results (1.0) for most projects. Most of the graphs shown in this paper show absolutely no difference at all. We can therefore state that, for the studied projects, the behavior of the project is one of continuous release, where the evolution after the first stable release is the same than when the project was still considered not to be release-quality.

In any case, the studied systems show a growth rate that is higher than the smooth growth one and in concordance with the findings of Succi et al. for GCC and Apache [22]. This could mean that the Fourth Lehman Law of Evolution does not apply to these large libre software systems (although our analysis has some differences with a classical Lehman study, which should be further researched).

If this were the case, it could be a consequence of the particular allocation and availability of human resources in libre software projects with large surrounding communities. For instance, there are many tasks (such as testing and bug reporting) that are in fact performed outside the core group of developers, which means that man power in some sense external to the project is actually actively collaborating to its growth. In other words, the flexible and auto-organized management of human resources usually found in those projects could be the reason of a growth rate higher than the one found in other, more rigid and planned cases.

All this said, there is still not enough evidence to state that linear growth is common in (large) libre soft-

ware systems, and even if that were the case (as it seems from our study), why that happens, and to which extent that contradicts Fourth Lehman's Law. Therefore, more projects should be studied, to improve the evidence (or find cases where growth is not linear), and detailed analysis should be performed of how human resources are allocated in libre software projects, with the aim of explaining the linear growth we have found.

8. Acknowledgments

We thank Juan Antonio Almendral, from the Mathematics, Physics and Natural Sciences Department of the Universidad Rey Juan Carlos for his invaluable help with the statistics of this paper.

References

[1] E. Burd and M. Munro. Evaluating the evolution of a C application. In *Intl Workshop on Principles of Software Evolution*, Fukuoka, Japan, June 1999.

[2] A. Capiluppi. Models for the evolution of os projects. In *Proceedings of the Intl Conf on Software Maintenance*, pages 65–74, Amsterdam, The Netherlands, 2003.

[3] A. Capiluppi, M. Morisio, and P. Lago. Evolution of understandability in oss projects. In *Proceedings of the 8th European Conf on Software Maintenance and Reengineering*, Tampere, Finland, 2004.

[4] T. Dinh-Trong and J. M. Bieman. Open source software development: A case study of freebsd. In *Proceedings of the 10th Intl Software Metrics Symposium*, Chicago, IL, USA, September 2004.

[5] J. R. Ehrenkranzt. Release management within open source projects. In *Proceedings 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, 2003.

[6] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the Intl Conf on Software Maintenance*, pages 23–32, Amsterdam, The Netherlands, September 2003.

[7] H. Gall, M. Jazayeri, R. Klösch, and G. Trausmuth. Software evolution observations based on product release history. In *Proceedings of the Intl Conf on Software Maintenance*, pages 160–170, 1997.

[8] D. Germn. The GNOME project: a case study of open source, global software development. *J of Softw Process: Improvement and Practice*, 8(4):201–215, 2004.

[9] M. Godfrey and Q. Tu. Evolution in Open Source software: A case study. In *Proceedings of the Intl Conf on Software Maintenance (ICSM 2000)*, pages 131–142, San Jose, California, 2000.

[10] M. Godfrey and Q. Tu. Growth, evolution, and structural change in open source software. In *Intl Workshop on Principles of Software Evolution*, Vienna, Austria, September 2001.

[11] J. M. Gonzalez-Barahona, M. A. Ortuo Perez, P. de las Heras Quiros, J. Centeno Gonzalez, and V. Matellan Olivera. Counting potatoes: the size of Debian 2.2. *Upgrade Magazine*, II(6):60–66, Dec. 2001.

[12] J. M. Gonzalez-Barahona, G. Robles, M. Ortuo Prez, L. Rodero-Merino, J. Centeno-Gonzalez, V. Matellan-Olivera, E. Castro-Barbero, and P. de-las Heras-Quirs. Analyzing the anatomy of GNU/Linux distributions: methodology and case studies (Red Hat and Debian). In S. Koch, editor, *Free/Open Source Software Development*, pages 27–58. Idea Group, Hershey, PA, 2004.

[13] K. Healy and A. Schussman. The ecology of open-source software development. Technical report, University of Arizona, USA, January 2003.

[14] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[15] M. Lehman and J. F. Ramil. Rules and tools for software evolution planning and management. *Annals of Software Engineering*, 11(1):15–44, 2001.

[16] M. Lehman, J. F. Ramil, and U. Sandler. An approach to modelling long-term growth trends in software systems. In *Intl Conf on Software Maintenance*, pages 219–228, Florence, Italy, November 2001.

[17] M. Lehman, J. F. Ramil, P. Wernick, and D. Perry. Metrics and laws of software evolution - the nineties view. In *Proceedings of the Fourth Intl Software Metrics Symposium*, Portland, Oregon, 1997.

[18] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[19] J. W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *Transactions on Softw Eng*, 30(4), April 2004.

[20] G. Robles, J. M. Gonzalez-Barahona, and R. A. Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In *Proceedings of the Intl Workshop on Mining Software Repositories*, Edinburg, Scotland, UK, 2004.

[21] S. R. Schach, B. Jin, D. R. Wright, G. Z. Heller, and A. J. Offutt. Maintainability of the linux kernel. *IEE Proceedings–Software*, 149:18–23, 2002.

[22] G. Succi, J. Paulson, and A. Eberlein. Preliminary results from an empirical study on the growth of open source and commercial software products. In *EDSER-3 Workshop*, Toronto, Canada, May 2001.

[23] W. M. Turski. Reference model for smooth growth of software systems. *IEEE Transactions on Software Engineering*, 22(8):599–600, 1996.

[24] T. Yamamoto, M. Matsushita, T. Kamiya, and K. Inoue. Measuring similarity of large software systems based on source code correspondence. In *6th Intl PROFES (Product Focused Software Process Improvement) conference, PROFES 2005*, Oulu, Finland, June 2005.