

# Preliminary Results from an Empirical Study on the Growth of Open Source and Commercial Software Products

## Giancarlo Succi

Department of Electrical and  
Computer Engineering  
University of Alberta  
Edmonton, Alberta, Canada  
T6G 2G7  
1-780-492-4992  
Giancarlo.Succi@ee.ualberta.ca

## James Paulson

Novatel Wireless  
Technologies, Ltd.  
Suite 200, 6715-8<sup>th</sup> Street NE  
Calgary, Alberta, Canada  
T2E 7H7  
1-403-295-4859  
jppaulson@novatelwireless.com

## Armin Eberlein

Department of Electrical and  
Computer Engineering  
University of Calgary  
Calgary, Alberta, Canada  
T2N 1N4  
1-403-220-5002  
eberlein@enel.ucalgary.ca

## ABSTRACT

This article describes an empirical study of the growth and evolution of three open source and three commercial software projects. This study identifies commonality and differences in the growth of these projects, and discusses how these results may be useful in planning software development strategies.

## Keywords

Software growth, evolution, open source, empirical studies

## 1 INTRODUCTION

The lives of “open source” and of commercial software products often span several years, and they evolve over several versions, often with increased functionality. Since the pioneering work of Lehman [3], several models have been proposed to describe such evolution. Often companies plan explicitly the introduction of new functionality in a software product. This has several advantages, such as:

- Reduction of the upfront costs required to develop the product.
- Mitigations of the risk involved in creating new products from scratch, by determining features the users are most interested in and ready to pay for.
- Establishment of an installed base of software, which can then be used to exploit that products or compatible products through network externalities.

A full understanding of the economic principles behind the

incremental introductions of functionality in software products is still missing. In particular, it is interesting to analyse how commercial and open source software products grow.

An appropriate characterization of this phenomenon provides the ground to support two possible strategies to plan the incremental development of software products:

- The use open source approaches to establish new, or penetrate closed existing markets.
- The partnerships between firms and with government agencies to establish new economically viable frameworks that can then be the backbone for their own products.

There are already examples of these strategies into action. For instance, Sun Microsystems has decided to make available to the general public most of the Java environment. The government of Georgia has established the “Yamacraw” project to support the establishment of a development framework to be shared by several partnering companies in the domain of microchip design.

Attempts have been made to model the incremental introduction of software products. Sullivan et al. [5], Erdogmus [2], and others have proposed to use the theory of real options to analyse the trade-offs between prototyping and developing a fully featured product.

This paper describes an empirical investigation on the software evolution of three commercial and of three open source software systems. The goal was to compare the evolutionary characteristics, determine the presence of any similarities or differences, and to determine any guidelines to aid in the software development process.

The commercial systems analyzed were software products from Novatel Wireless, a provider of wireless Internet data products and services. The three products analyzed were the embedded software core of three different wireless

protocol products. They have been identified as Project A, B, and C to honor confidentiality agreements. The open source systems investigated were three well-known open source projects: Linux operating system, Apache Web Server, and the Gnu Compiler Compiler (GCC). For comparative purposes, all of the projects chosen were written in the C programming language.

## 2 DATA ANALYSIS

Our research data collection and analysis involved using an automated tool to capture metrics from a source code snapshot of each release of the project, and then generating project metrics by comparing subsequent releases. The metrics that were captured and calculated can be generalized into the following categories:

- **Release Timing Metrics:** Metrics related to the date of the release, the release sequence number, and the time in between releases.
- **Function Count Metrics:** The total number of C files, the total number of functions in the release, and the number of functions added or modified.
- **Size Metrics:** The total lines of code of all the functions in the release, the total lines of code of the functions added, the average lines of code of an added function, and similar metrics for modified functions.
- **Complexity Metrics:** The total complexity of all the, the total complexity of the functions added, the average complexity of an added function, and similar metrics for modified functions.
- **Usage / Reuse Metrics:** The percent of functions that were reused from previous releases, the static count of the number of times a function is used in a release and subsequent releases, the average number of times that an added function is reused in subsequent releases, and the change in how a function is used and reused in subsequent releases.

These metrics were captured and calculated across all the releases for a particular project. The data was normalized for comparison using two different methods. The first method was by calculating the metric value as a percentage of the final release metric value. This allows a comparative graph of all the projects that converge at 100% for the final release value, and makes a visual interpretation of the data easier. An example of this is shown in Figure 2.

The second method was performed on certain metrics to aid in determining a relative comparison. The data was normalized by calculating the metric value as a percentage of the total metric value. For example, a calculation of the total lines of code of the functions that were added as a percentage of the total lines of code in the entire release was performed. An example of this is shown in Figure 4.

The metrics were also compared and analyzed against two different horizontal axes. The project metrics were compared against the release sequence number (RSN) as defined by Cox and Lewis [1]. They were also examined against the number of days from the initial release. This allowed a comparison of the relationships between both the order and the timing of the releases.

### Correlation Analysis

The second part of the analysis involved calculating the correlation for each metric across all projects. Correlation for both the RSN horizontal axis and the release time axis was performed. Correlation measures the relationship between two data sets that are scaled to be independent of the unit of measurement. The population correlation calculation returns the covariance of two data sets divided by the product of their standard deviations. The release time axis had to be scaled to be independent by using a straight-line interpolation of the known data points to generate the data set.

The correlation gives a matrix of the correlation between each of the six projects. We extended the analysis by determining the average correlation between open source and other open source projects, between commercial and other commercial projects, and between open source and commercial projects. This allowed us to determine whether the data supported or rejected the hypothesis that open source projects had similar evolutionary characteristics of the metric being analyzed to that of commercial projects.

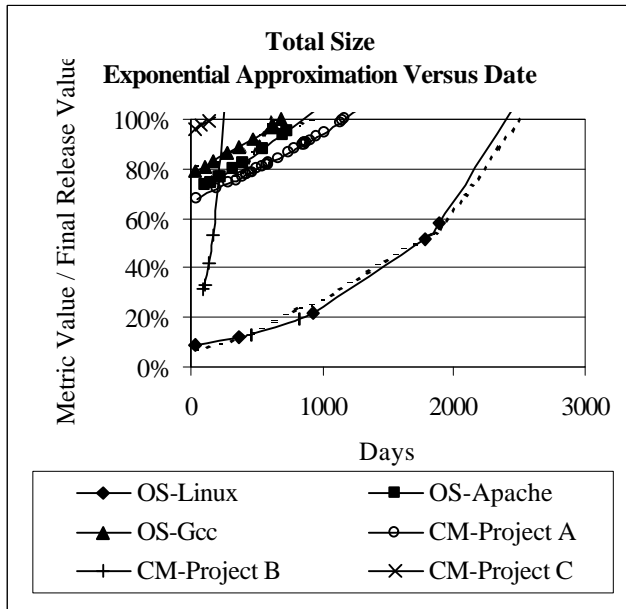
### Linear and Exponential Approximation

The third part of the analysis involved determining a linear and exponential approximation of the metric relationship. This yielded a comparison graph of all the projects for the metric. In addition, the equations for the graph were analyzed to see if there were any relationships between the equation parameters (m,b) for open source projects and that of commercial projects.

## 3 RESULTS AND DISCUSSION

The characteristics of the total functions, size, and complexity seem to indicate similar patterns. The graph in Figure 1 shows the size growth as a percentage of the final release value graphed as function of time. When we view the graphs individually, we see that an exponential approximation seems to be the best fit for the Linux project (which coincides with the findings of other researchers [4]). However, the other projects including the two other open source projects analyzed seem to support a more linear approximation of the growth. These results are the same when the complexity and the function count metrics are analyzed. This can also be verified by taking an exponential approximation of all the curves. When we examine the curves and the equations for the graph, we see that the OS Linux Project has a high m value in the

exponential equation  $y=b*m^x$ , whereas the other curves have an  $m$  value close to 1, indicating that a linear approximation is more appropriate for these curves.



**Figure 1: Exponential Approx. of the Total Size Growth**

There are a number of hypotheses that could explain these results:

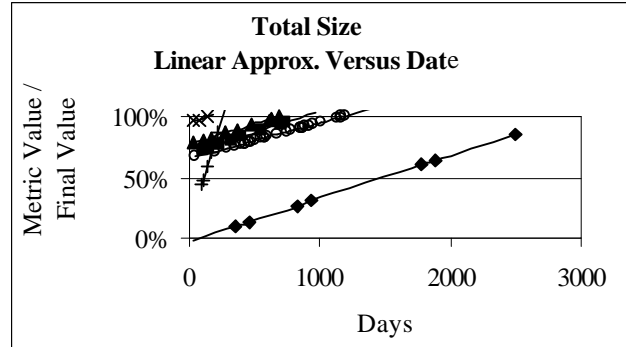
- We may be looking at snapshots of the releases in the flatter, more linear part of the exponential curve for the other five projects.
- We may be analyzing the projects over too small of a window to see the exponential growth.
- The two other open source projects we examined (Gcc and Apache) do not exhibit the high exponential growth we expect to see in open source projects.

In any case, it suggests that comparing the linear approximation of growth rates may be more useful in comparing open source and commercial projects than by comparing the exponential approximation.

**Linear Approximation of the Size, Complexity, and Function Count Metrics**

The majority of the open source and commercial projects analyzed had a similar growth rate with respect to the increase of the total functions, the total lines of code, and the total complexity when analyzed using linear approximation. A representative graph of the size is shown in Figure 2. This relationship is similar to that found for the total complexity and the total number of functions. In all cases, the relationship was not as clear when compared against the release sequence number. Project B is the

anomaly and has a much higher change over time than all the other projects. However this can be since the first two releases were early non-functional engineering releases, and the third release was a functional releases for the product. This sharp jump skews the linear and exponential approximations and the rate of change is much lower when examined further out on the horizontal axis.



**Figure 2: Linear Approx. of the Total Size**

The correlation factors for both the graphs also show high positive correlation across all the projects, although there is higher correlation between open source than commercial projects, but this is attributable to commercial Project B.

Average Correlation	OS to OS	OS to CM	CM to CM
Size – Date Correlation	0.86	0.82	0.65
Size – RSN Correlation	0.91	0.68	0.50
Complexity – Data Correlation	0.88	0.82	0.47
Complexity – RSN Correlation	0.91	0.67	0.63

**Table 1: Average Correlation for the Total Size and Complexity**

We can take the average slope and y intercept, and calculate the linear equations for size, complexity, and total function growth. This gives us the following equations:

Size Growth:  $y=0.00033x+ 0. 61$

Complexity Growth:  $y=0.00031x + 0.61$

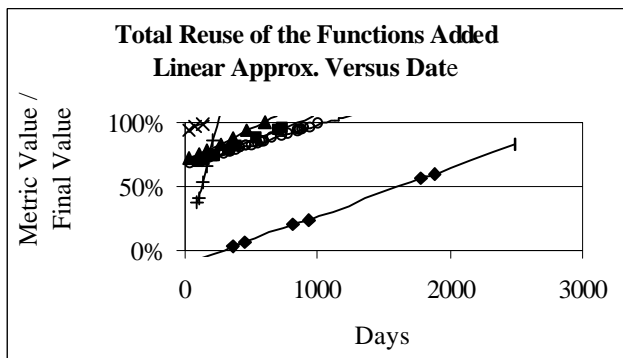
Function Growth:  $y=0.00028x + 0.59$

These equations can then be used to predict future growth of the projects in our domain, and can be compared against the growth rates of other domains. The slope and the y intercept also show that the growth of size, complexity, and number of functions are very similar. This relationship is also shown in examining the characteristics of the functions added, and the increased size and complexity of the functions added. This close relationship would appear to indicate that the rate of change of the number of functions

drives the change in size and complexity of the project. This is logical if the average size and average complexity of the functions that are added are reasonably constant throughout the lifecycle of the project. This is true as is shown in Table 2 for the average size and complexity of the functions added. Therefore examining the growth of the projects by one of the metrics (i.e. Total Size) is a reasonable approximation for the increase in complexity and total functions for the projects analyzed.

### Usage / Reuse Metrics

The graph in Figure 3 describes the total reuse of the functions added as a percentage of the final release value. The total reuse is calculated by counting the number of times a function is used in a release. Although it does not represent the actual dynamic program flow, it is a useful approximation of the dynamic behavior of the system. Figure 3 shows that a linear approximation gives a similar slope across both open source and commercial projects, with the exception of Project B.



**Figure 3: Linear Approx. of the Total Reuse of the Functions Added**

This would indicate that the rate of change in the use/reuse of functions is similar for both open source and commercial projects. We can determine a linear equation for the average of the lines to be:

$$y = 1.00069x + 0.60$$

The fact that the slope is greater than the rate of change for function count, size, and complexity graphs examined earlier indicates that reuse plays a large part in the evolution of software, and that both open source and commercial projects seem to be affected similarly. This is contrary to what you would expect if the open source projects analyzed were designed more modular than commercial projects.

### Functions Added

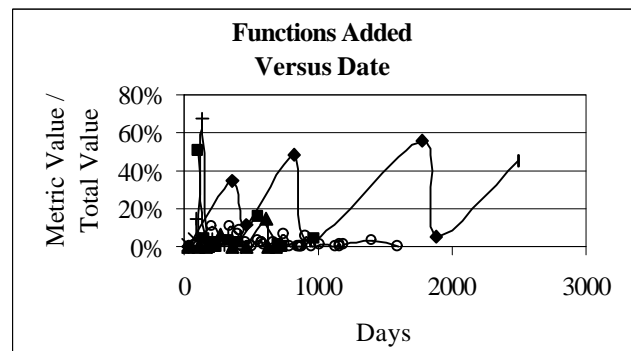
The table below indicates that the average size, complexity, and reuse of the functions that are added in open source projects are greater than that of commercial projects.

Proj cate	Size		Complexity		Reuse	
	Avg	Stdev	Avg	Stdev	Avg	Stdev
ALL	14.3	4.2	7.3	2.0	92.8	196.3
OS	16.2	3.5	9.1	3.7	180.5	270.6
CM	12.5	3.5	5.5	1.7	5.1	0.3

**Table 2: Average Size and Complexity of the Functions Added**

This indicates that on average, the functions added in open source projects are larger and more complex than the functions added in commercial projects. This also shows that functions are more widely used in open source projects than in commercial projects, and could indicate the flaws of only doing a static analysis of the code to measure growth. The count of the number of times a function is used/reused gives a more accurate picture of the total impact that adding a function has on future releases.

There was not a high degree correlation between the total number of functions that were added or modified in each release for the projects. However, the graph in Figure 4 shows that the number of functions added as a percentage of the final release value and as percentage of the current release total value seems to fluctuate cyclically.



**Figure 4: Functions Added**

In addition, the graphs of the increase in size and complexity due to the functions that are added or modified closely matched the graph of the functions added itself. These graphs all demonstrated a cyclical pattern as shown in Figure 4.

## 4 CONCLUSIONS AND FUTURE RESEARCH

The analysis of the data to date has yielded the following conclusions:

- A linear approximation represents the growth rate of the majority of open source and commercial projects analyzed, with the exception being the Linux open source project.

We predicted that the open source projects would exhibit exponential growth, and that commercial projects would exhibit more linear growth. Our analysis has not shown this to be the case. Linear growth models seem to best represent the majority of all the projects analyzed.

- *Open source and commercial projects have a similar growth rate over time with respect to the total functions, the total size, and the total complexity of the release.*

It is interesting that the growth rates of both open source and commercial projects are similar with respect to time. We would have expected that the percentage growth rate over time for open source projects would have been much greater than that of commercial projects even with a linear approximation. The only anomaly in the analysis is for Project B, a commercial project whose growth rate was even higher than that of other commercial and open source projects.

- *Open source and commercial projects have similar growth rate over time with respect to the total reuse of the functions added. This growth rate is higher than the growth rate for total functions, size, and complexity growth.*

This finding is important, as it confirms that a statistical analysis of the evolution is not a complete picture by itself, and that an analysis of the reuse and program flow is important in understanding how projects evolve. It is again surprising that the growth rate for the reuse of the functions added over time is similar for both open source and commercial projects. One would generally have expected more total reuse and reuse growth for the more modular open source projects than for commercial projects.

- *The total number, total size, and total complexity of the functions added is generally cyclical in nature for both open source and commercial projects*

Lehman [3] notes that this cyclic pattern “is characteristic of self-stabilizing feedback systems”. Although this pattern was not as clearly visible in the analysis of all the functions (rather than just the functions added), it is clearly visible in most of the graphs relating to the number, size, and complexity of the functions added.

- *The average size, the average complexity, and the average reuse of a function added are generally greater for open source than for commercial projects.*

This finding indicates that the functions added for open source projects are generally larger and more complex, and are more widely reused than in commercial projects. One theory to explain this result is that there may be less control and review of the modules submitted in open source projects, whereas commercial projects may have more structured guidelines on a function’s size and complexity. Another theory could be that the open source projects examined (an operating system, a web server, and a compiler compiler) may be different than the commercial projects examined (three embedded protocol stack cores). The nature of the projects themselves may have defined the characteristics of the functions added.

Generally, we found more relationships supporting similar evolutionary characteristics between open source and commercial projects. This could indicate that other external factors have more impact on the evolutionary characteristics of the software than the economic model for the production of the software itself.

Further research could involve comparing this analysis to other open source and commercial projects for comparison. In addition, further exploration into the reuse and dynamic characteristics of evolutionary characteristics could aid in understanding the similarities and differences in open source and commercial software.

#### **ACKNOWLEDGEMENTS**

We gratefully acknowledge the contribution of Novatel Wireless Technologies Ltd. for access to the commercial archival source code.

#### **REFERENCES**

1. Cox, D.R., and Lewis, P.A., (1966), *The Statistical Analysis of Series of Events*, John Wiley and Sons, New York, New York.
2. Erdogmus H. (1999) "Valuation of complex options in software development", *First Workshop on Economics-Driven Software Engineering Research*, Los Angeles, CA.
3. Lehman, M.M. (1996) "Feedback in the software evolution process", *Information and Software Technology*, vol.38, pp.681-686.
4. MacCormack, A. "How Internet Companies Build Software", *MIT Sloan Management Review*, Winter 2001.
5. Sullivan, K.J., P. Chalasani, S. Jha and V. Sazawal, (1999) "Software Design as an Investment Activity: A Real Options Perspective," *Real Options and Business Strategy: Applications to Decision Making*, L. Trigeorgis, consulting editor, Risk Books