An Empirical Study of Open-Source and Closed-Source Software Products

James W. Paulson, *Member*, *IEEE*, Giancarlo Succi, *Member*, *IEEE Computer Society*, and Armin Eberlein, *Member*, *IEEE Computer Society*

Abstract—This paper describes an empirical study of open-source and closed-source software projects. The motivation for this research is to quantitatively investigate common perceptions about open-source projects, and to validate these perceptions through an empirical study. This paper investigates the hypothesis that open-source software grows more quickly, but does not find evidence to support this. The project growth is similar for all the projects in the analysis, indicating that other factors may limit growth. The hypothesis that creativity is more prevalent in open-source software is also examined, and evidence to support this hypothesis is found using the metric of functions added over time. The concept of open-source projects succeeding because of their simplicity is not supported by the analysis, nor is the hypothesis of open-source projects being more modular. However, the belief that defects are found and fixed more rapidly in open-source projects is supported by an analysis of the functions modified. The paper finds support for two of the five common beliefs and concludes that, when implementing or switching to the open-source development model, practitioners should ensure that an appropriate metrics collection strategy is in place to verify the perceived benefits.

Index Terms—Open source, software metrics, empirical studies, software growth models.

1 Introduction

The use of the open-source development method as a viable alternative to closed-source development is of particular interest to the software industry today. Several companies have switched from a closed source to an open-source development model in an effort to win market share, to expand product growth, and improve market penetration [8]. This strategy is often based on anecdotal evidence surrounding the characteristics of open-source growth rather than on any empirical studies of how open-source and closed-source projects grow and evolve over time. An appropriate investigation of these perceptions provides companies with the evidence needed to justify the use of open-source software as an alternative to closed-source software. Wheeler [14] provides quantitative data with respect to market share, reliability, performance, scalability, security, and total cost of ownership of open-source projects as a whole. The aim of this paper is to compare the evolutionary and static characteristics of open-source and closed-source software, to examine the claims about open-source software, and to discuss how these results may influence software development strategies. To achieve this goal, the paper describes an empirical investigation of three closed-source and three open-source software systems.

- *J. Paulson is with General Dynamics Canada Ltd., 1020—68th Avenue, N.E. Calgary, Alberta, Canada T2E 8P2. E-mail: jim.paulson@ieee.org.*
- *G. Succi is with the Center for Applied Software Engineering, Faculty of Computer Science Free University of Bozen, Dominikanerplatz 3 Piazza Domenicani, I-39100 Bozen, Italy. E-mail: Giancarlo.Succi@unibz.it.*
- *A. Eberlein is with the Computer Engineering Department, American University of Sharjah, PO Box 26666 Sharjah, United Arab Emirates. E-mail: eberlein@enel.ucalgary.ca.*

Manuscript received 17 Jan. 2003; revised 25 Aug. 2003; accepted 4 Feb. 2004. Recommended for acceptance by P. Johnson.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118154.

This paper is organized as follows: Section 2 starts with a revision of the work performed in this area. Section 3 discusses the research design and presents the reference systems used in the analysis. Section 4 provides a discussion of the results with the conclusions summarized in Section 5 along with directions for future research.

2 State of the Art

Examining the lifecycle and characteristics of open-source systems is important to understand and explains why open-source systems may evolve differently than closed-source systems. The following section describes the statements and references made regarding open-source software development as found in the literature survey and also describes the metrics that were chosen to quantitatively investigate these statements.

Open source development fosters faster system growth. The concept of open-source systems exhibiting strong growth is confirmed by the study of Mockus et al. [7]. System growth is estimated and compared by using the metric of percentage growth over time. Using the release sequence number (RSN) is not a valid metric for project comparison since deciding when to release is arbitrary. We estimate software growth as a percentage by taking the total source lines of code in the project, and dividing it by the total source lines of code in the final release analyzed for the project. We then graph this information against the calendar time of each release. This approach was also used by Cook and Roesch [1] in their analysis as a method of comparing project growth.

Open source projects foster more creativity. O'Reilly [9] and Dalle and Jullien [2] note that creativity is more prevalent in open-source systems, which leads to more rapid development of user features. We attempt to estimate creativity by

examining the number of features or functions added per release. This is a reasonable metric for quantitatively measuring creativity, since only the functions added are counted, not the functions modified. The assumption made is that, in order to add new features to a software project, you must add a relative number of functions. The limitation of this assumption is that the number of modules added to support a single new feature is not constant, but it does provide a reasonable metric for comparison of creativity.

Open source projects succeed because of their simplicity. Raymond [12] notes that open-source systems often succeed because of their simplicity. If open-source systems are generally written simpler than closed-source systems, we expect to see the average complexity of the modules added for open source to be less than that of closed-source systems. The limitation of using this metric as an indicator of the system complexity is that both Raymond [12] and Cook and Roesch [1] note that the complexity of some systems may be found in its data structures rather than in its source code. Unfortunately, metrics for the data structures of the code was not captured in our research because of the limitation of our metrics collection tool.

Open source projects generally have fewer defects than closed-source projects since defects are found and fixed more rapidly. Mockus et al. [7] notes that defects are detected and fixed faster than in closed-source projects. This statement is confirmed by Wheeler [13] who states that the open-source community was more responsive in identifying and fixing defects. This results in improved reliability in open-source projects. Our analysis uses the metric of the functions modified over time to quantitatively examine this statement. This is also referred to as the changing rate. We hypothesize that open-source systems should exhibit higher changing rates over time, since defect fixes would result in more functions modified to address the large number of defects found. The use of this metric has several limitations. It assumes that the number of functions modified per defect is constant and proportional across projects, which is not likely the case. It also assumes that defect fixes only result in functions being modified, rather than functions added. Defects may require refactoring of the software, which will generate new functions in addition to modified functions. However, the changing rate is an adequate measure of determining when defects are fixed through a static analysis of the software.

Open source projects are more modular than closed-source projects. O'Reilly [9] notes that the extensibility of the system is a characteristic of open-source systems. If open-source systems are generally more extensible than closed-source systems, we would expect to find tighter coupling in closed-source systems. The addition of features in closed-source systems would require more changes in existing modules to support a new feature and might cause us to see more modules changed per code release in closed-source systems for every feature added. An examination of the correlation between the changing rate (modules modified) and growing rate (modules added) in closed-source projects compared to open-source projects may be a good indicator of this phenomenon. If there is strong correlation, this indicates that the changing rate is linked to the growing rate and may be an indicator that modules needed to be modified for every feature added and is a useful metric for project comparison.

The metrics chosen attempt to quantitatively compare open-source project characteristics to closed-source project characteristics. There are a number of limitations to this study. O'Reilly [9], Raymond [12], and Wheeler [13] all agree that open-source systems need to be in a fairly developed state to be successful. They suggest that the developers and users need to have something they can run and test in order to contribute to further open software development, and that systems that are started in a "bazaar" style are not likely to succeed [13]. The author refers to the "bazaar" style approach to software development as a great babbling bazaar of differing agendas and approaches where submissions are taken from anyone. If open-source systems need to be relatively developed, this implies that both open-source and closed-source systems begin in a similar manner and then deviate once the source code is released and the development takes on a certain critical mass. This suggests that the characteristics of open-source systems might be dependent on what phase of the lifecycle that the open-source system is currently in. If open-source and closed-source systems begin in a similar manner, then examining the growth of either project early in its lifecycle may result in viewing similar growth patterns. It is difficult to identify common phases when comparing several different projects. All of the closed-source projects in the analysis were analyzed well into the maintenance phase, but the data included the early developmental releases as well. For the open-source projects, the analysis of Linux uses only the major developmental releases for project comparison. Since each major developmental release takes into account all of the previous developmental releases, we felt that analyzing these releases gives an accurate comparison to how the closed-source projects were developed at the expense of the granularity sacrificed by omitting some of the developmental releases. Godfrey and Tu [4] investigate the evolution of Linux with regard to development and stable releases. Our analysis focuses on only the kernel portion of Linux since this is most comparable to the closed-source projects in our analysis and augments the work of Godfrey and Tu with additional metrics including the growing and changing rate.

The results cannot be said to be indicative of all open-source and closed-source projects. Godfrey and Tu [4] and Lehman [6] note that there are a number of factors influencing how software evolves or grows over time. It is difficult to quantify and describe all of the factors that may affect how software grows, but a number of proposed sources include the application domain, the software development process, the type of product, the management, marketing dynamics, and organizational dynamics. Our results are not intended to prove nor disprove the anecdotal claims made by open-source software community, but to investigate those claims using the projects in our study and to provide a basis for other researchers to add to this body of knowledge.

TABLE 1
Hypotheses and Metrics

Current Research:	Reference:	Metric:
Open source development fosters faster system growth.	Wheeler [13] O'Reilly [9] Raymond [12] Mockus *et al.* [7] O'Reilly [9] Dalle and Jullien [2]	Overall project growth in functions over time.
		Overall project growth in LOC over time.
		Overall project growth in complexity over time.
Open source projects foster more creativity.	Wheeler [13] O'Reilly [9] Raymond [12] Mockus *et al.* [7] O'Reilly [9] Dalle and Jullien [2]	Functions added over time.
Open source projects succeed because of their simplicity.	Raymond [12]	Overall project complexity.
		Average complexity of all of the functions.
		Average complexity of the functions added.
Open source projects generally have fewer defects than closed source projects, as defects are found and fixed rapidly.	Mockus *et al.* [7] O'Reilly [9]	Functions modified over time.
		Functions modified as a percentage of total functions.
Open source projects are more modular than closed source projects.	Mockus *et al.* [7] O'Reilly [9]	Correlation between functions added and functions modified.

3 Research Design and Reference Systems

Kemerer and Slaughter [5] note that one of the most difficult aspects of an empirical study of software growth is to find a good closed-source partner that has the archives and data necessary and who is willing to give researchers access to proprietary source code. This was ensured in our research by using a closed-source partner who provided us access to the source code archives of three closed-source projects. We also use the open-source archives of three readily available and widely used open-source systems whose installed base is similar to that of very successful closed-source software. Kemerer and Slaughter also note that another key success factor is that the empirical research requires a disciplined and structured approach to data collection due to the enormity of the data to be collected. We met this criterion by developing a metrics plan based on a Goal Question Metric method, which indicates specifically the metrics to capture, and the goals of the research that they address. The metrics that we chose to validate the hypotheses derived in the previous section are shown in Table 1. The rationale for choosing the metrics was based on validating the hypotheses mentioned previously. The metrics we chose are all used in previous empirical studies by other researchers so that our data provides a meaningful comparison. A complete list of all the metrics recorded or calculated by the tool can be found in [10], along with the goals they address.

The reference systems chosen are three closed-source projects and three widely known open-source projects. The three closed-source systems analyzed are the embedded software core of three different wireless protocol products. In order to honor confidentiality agreements, they have been named as Project A, Project B, and Project C. Although the domain of the three projects is the same, the projects themselves are considerably different as is demonstrated by the description of the characteristics. The majority of the code for all three projects is developed and purchased from three different closed-source partners. For most of the code analyzed for the three closed-source projects, none of the developers, processes, and programming techniques were common. The lack of commonality in characteristics and in the design will add validity to any commonalities found in the data analysis. The closed-source projects are identified in the analysis as "CS."

Three well-known open-source projects are chosen to represent open-source development. Linux is an operating system that was initially created as a hobby by a student, Linus Torvalds, at the University of Helsinki in Finland. GCC stands for GNU Compiler Collection and is an open-source C compiler that is portable to multiple architectures and environments. Apache is an open-source project that

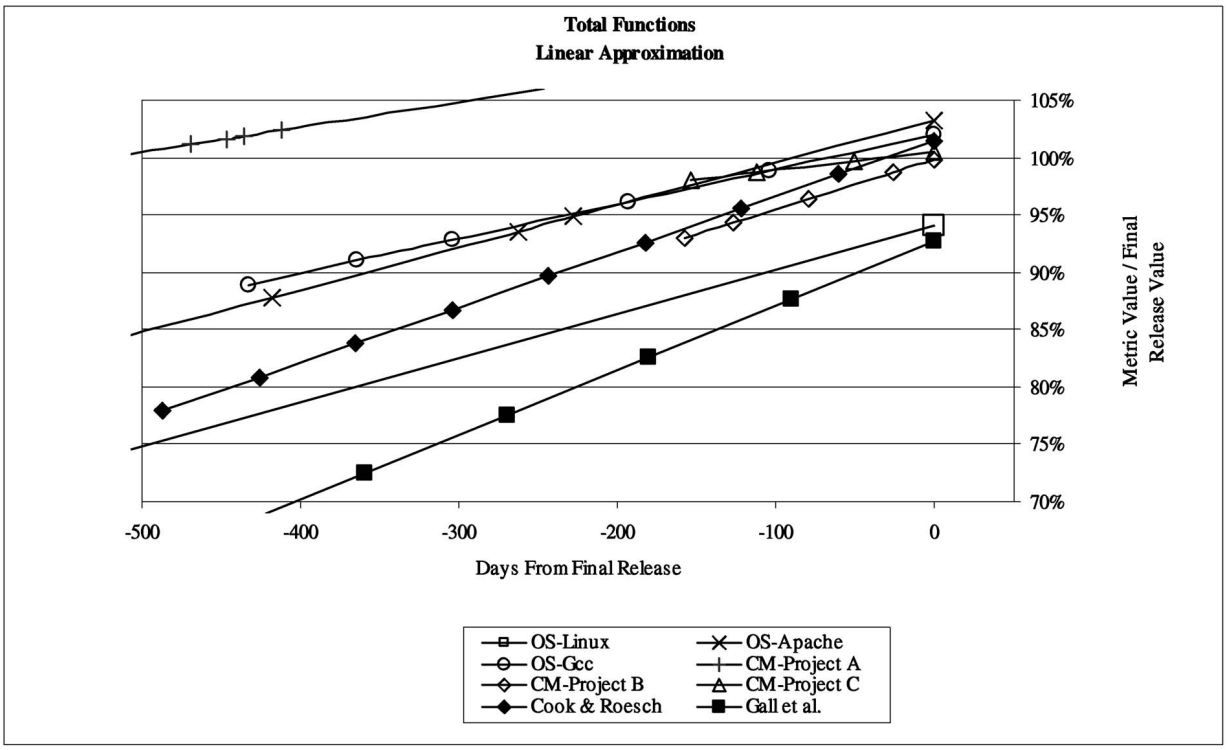

Fig. 1. Linear approximation of total project size.

was aimed at creating a robust, closed-source-grade, featureful, and freely available source code implementation of an HTTP Web server. The open-source projects are identified in the analysis as "OS."

It is important to note that the value of our research is dependent on how representative our projects are of open-source and closed-source software development. The three open-source projects Linux, Apache, and Gcc are the most well-known and successful examples of open-source projects. As such, they are a representation of projects that have been developed using open-source methods. The only negative aspect to choosing these projects might be their success. The growth characteristics we observe in these projects may be a result of their widespread adoption rather than their economic development method. The three closed-source projects we chose are representative of a particular closed-source domain and not of closed-source software in general. The domain of our closed-source projects can be described as the software protocol stacks of wireless telecommunications products. Our comparison may be focused on a specific closed-source domain, but this does not negate the validity of our conclusions. Instead, it provides a good baseline for researchers to compare and validate our findings on projects in other closed-source domains.

The data for these projects is determined using a tool we developed that steps through each source code archive release of a project, and determines both the release timing metrics such as the module growth over time, as well as the static metrics such as the average size and complexity of a function added in a release. This tool is described in further detail in [10].

4 Results and Discussion

4.1 Growth

The majority of the open-source and closed-source projects analyzed have a similar growth rate with respect to the increase of the total lines of code, the total complexity, and the total number of functions when analyzed using a linear approximation. A linear model was chosen based on an analysis of several different models [10] and provides a reasonable approximation of the growth for project comparisons. A representative graph of the total number of functions is shown in Fig. 1. This relationship is identical to that found for the growth of total complexity and the total size in LOC for the project. The linear model parameters for size growth are shown in Table 2. The parameters a and b are the coefficients of the linear equation $y = a + bx$. The parameter x is the time in days from initial release, and the parameter y is the percentage growth increase of the number of functions over time. Two parameters are calculated and shown in the table as well. The absolute sum of squares of the error of the linear model compared to the actual data is shown in the table, which indicates how well the linear model tracks the actual data. The Pearson Product Moment Correlation [11], r, calculated between the time in days and the actual percentage growth increase, reflects the extent of a linear relationship between the two sets.

If we compare the equations for the graphs of the linear approximation, we can see that the range of slopes for open-source projects given by the parameter b (0.00030 to 0.00039) is similar to that of closed-source projects (0.00016 to 0.00043). Similar patterns were seen for the slopes of the

TABLE 2
Linear Model Parameters for Project Size Growth

Project	Linear Model y = a + bx		Absolute Sum of the Squares Of the Differences	Pearson Product Moment Correlation Coefficient
	a	b		
OS-Linux	-0.129168	0.00039	0.0153	0.9847
OS-Apache	0.676096	0.00037	0.0064	0.9701
OS-Gcc	0.878866	0.00030	0.0030	0.8955
CM-Project A	0.762840	0.00022	0.0751	0.8443
CM-Project B	0.916006	0.00043	0.0002	0.9688
CM-Project C	0.974849	0.00016	0.0106	0.9330
Average:	0.679915	0.00031		
OS Average:	0.475265	0.00035		
CM Average:	0.884565	0.00027		
Cook & Roesch	0.749167	0.00048	0.0012	0.9888
Gall et al.	0.573538	0.00056	0.0106	0.9330

total complexity and total size growth. It is also important to note that the correlation coefficient shows all of the projects have strong positive correlation, indicating that a linear model is a good approximation for the behavior of the growth over time. These results suggest that open-source and closed-source 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 to compare our results against the findings of other researchers. In particular, the findings of Cook and Roesch [1] and Gall et al. [3] were compared. The graph in Fig. 1 includes a linear approximation of the LOC data provided by Cook and Roesch and that of Gall et al. Although the data published by these researchers uses release sequence numbers, an approximation was made of the days from initial release by averaging the time span of the data studied across the releases analyzed. We can see that the slopes of the lines for the data from Cook and Roesch [1] and from Gall et al. [3] fit into the pattern of slopes we see from our results. By calculating the equations for the size growth of these researchers' data, we can compare the slope or the size growth rate for each of the various projects. The slope of the linear approximation for the project growth over time of Cook and Roesch [1] and Gall et al. [3] is shown in Table 2 as well. All of the projects have a very similar slope or project growth over time. The similarity of the growth of project size over time to our analysis results may be explained by the fact that the projects analyzed by Cook and Roesch and Gall et al. have very similar characteristics to the closed-source projects in our analysis. The closed-source projects we examine are embedded real-time systems, which is the same as the project analyzed by Cook and Roesch. In addition, the projects analyzed by both Cook and Roesch and Gall et al. are telecommunication products. This is closely related to the domain of our closed-source projects. This suggests that the growth rate of software development with respect to time may not be entirely dependent on the economic model nor on the project itself, but may have more commonality throughout a particular domain. It would be interesting to compare these results with other domains to determine possible analogies.

The results indicate that open-source and closed-source projects have a similar project size growth rate over time with respect to the total number of functions and the total size in LOC. We expect that the percentage growth rate over time for open-source projects would be much greater than that of closed-source projects. The constant and similar growth rate between both closed-source and open-source

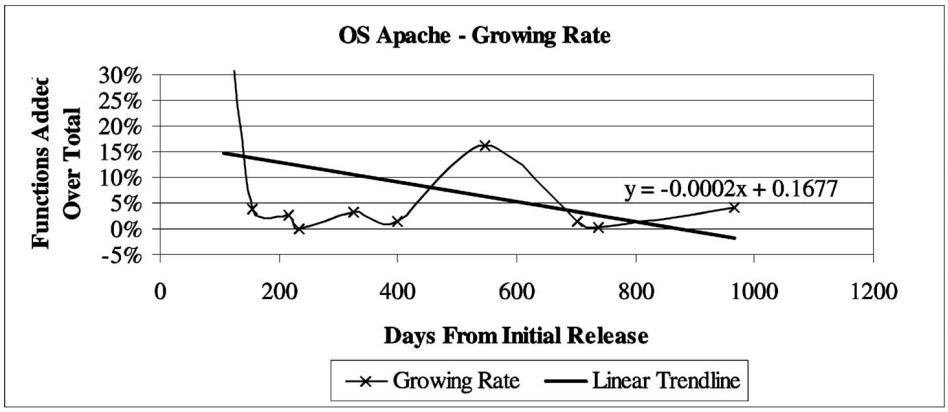

Fig. 2. Growing rate of apache over time.

projects is more clearly illustrated by the use of graphing the percentage increase in a metric value, rather than the metric value itself. Using this approach allows comparisons using the same scale and minimizes the impacts of the number of developers that a project has. Open source projects such as the Linux kernel with approximately 60,000 functions are significantly larger than some of the closed-source projects in our study and, therefore, the growth in LOC per release is larger. But, when normalized by examining the percentage growth, we get a value that is comparable for varying project sizes. While the number of developers is of concern, we are not interested in the personal productivity of closed-source and open-source projects, but rather on the overall growth of the projects.

Future research in this area could involve examining additional open-source and closed-source projects to see if the commonality in growth rates is related to the economic development model, or other project characteristics such as programming language or environment.

4.2 Creativity

Gall et al. [3] investigates the growing rate of projects. The growing rate is defined as the number of functions added over time. Gall et al. finds that the growing rate is cyclical, but generally decreases over time. A sample of the results for our projects using a similar analysis is shown in Fig. 2 and Fig. 3. Our results indicate that four out of the six projects analyzed (OS Apache, CM Project A, CM Project B, and CM Project C) have a decreasing growth rate. The exceptions are OS Linux and OS GCC, both of which have a positive growth rate. When we compare the slope of the linear trend lines of the open-source and closed-source projects in Table 3, we find that the average of the open-source projects is -0.000039, whereas the average of the closed-source projects in our study was -0.001031. This means that, on average, the open-source projects had a positive linear growth rate of approximately 26 times that of the closed-source projects. None of the commercial projects in the study show any positive growth rate over time. This supports the hypothesis that open-source projects foster more creativity, as the open-source projects in our study had higher positive growing rate than the closed-source projects. As mentioned previously, the success of the open-source projects in our study may have influenced the results, as well as the domain of the closed-source projects. However, comparing successful open-source projects to successful closed-source projects provides the grounds for a proper comparison.

4.3 Simplicity

We examine the complexity of the projects using three metrics. The total complexity of each of the project releases

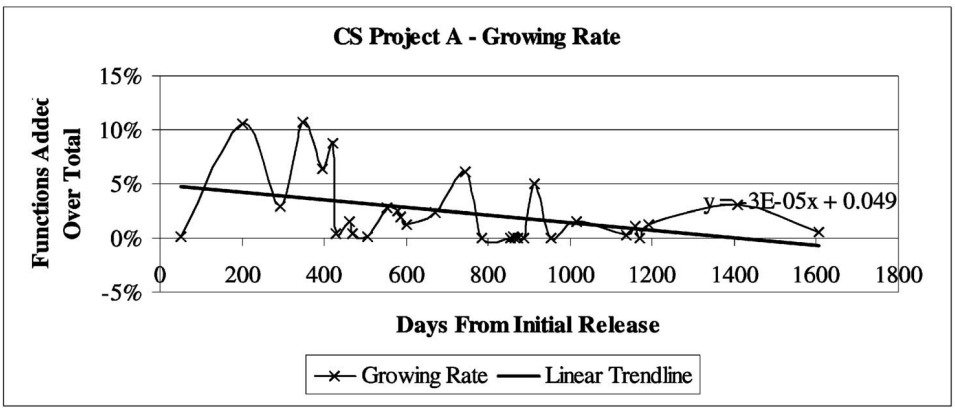

Fig. 3. Growing rate of Project A over time.

TABLE 3
Linear Model Parameters for Functions Added

Project	Linear Model y = a + bx	
	a	b
OS-Linux	0.410700	0.000058
OS-Apache	0.167720	-0.000193
OS-Gcc	0.006560	0.000018
CM-Project A	0.048963	-0.000035
CM-Project B	0.490304	-0.002934
CM-Project C	0.029668	-0.000125
Average:	0.192319	-0.000535
OS Average:	0.194994	-0.000039
CM Average:	0.189645	-0.001031

is examined and an average of all the releases performed. The second metric is calculated by dividing the total release complexity by the number of functions in a release and calculating the average function complexity across all the releases. The third metric involves analyzing only those metrics that were added in each release, determining the average complexity of an added function in a release by dividing by the complexity of the added functions by the number of added functions, and calculating an average across all the releases. The results of this analysis are shown in Table 4. For all three metrics, open-source projects have a higher complexity than the open-source projects. The total complexity for open-source projects is higher, the average complexity of a function is higher in open-source projects, and the average complexity of a function added in a release is higher in open-source projects compared to closed-source projects. Our results do not support the hypothesis that open-source projects are generally less complex than closed-source projects. One possible explanation for our results is that the domain of the closed-source projects was embedded software kernels, whereas the domain of the open-source projects was operating system and application software. Embedded software is often constrained by memory and size limitations, so the functions tend to be less complex to enhance real-time performance. Complexity is often strongly influenced by the individual programmer's style, as well as organizational processes that limit the complexity of an individual module as part of a coding standard. None of the closed-source projects in our analysis were subject to limitations on individual module complexity and, to our knowledge, none of the open-source projects were either. It would be interesting to compare the complexity of other domains to determine if there is a strong relationship between the development model and the complexity of the software.

4.4 Defects

The amount of defects is analyzed by examining the number of functions that changed over time. Gall et al. [3] refer to this as the changing rate. If defects are being found

TABLE 4
Complexity

Project	Average Release Complexity	Average Complexity of A Function	Average Complexity Of An Added Function
OS-Linux	193,821.40	9.83	9.87
OS-Apache	15,405.55	10.49	9.57
OS-Gcc	144,791.71	9.12	6.22
CS-Project A	7,555.78	5.21	4.28
CS-Project B	25,548.17	7.02	6.15
CS-Project C	17,143.40	7.60	5.83
Average:	67,377.67	8.21	6.99
OS Average:	118,006.22	9.81	8.55
CS Average:	16,749.12	6.61	5.42

TABLE 5
Linear Model Parameters for Functions Modified

Project	Linear Model y = a + bx	
	a	b
OS-Linux	0.228815	-0.000037
OS-Apache	0.128144	-0.000115
OS-Gcc	0.010065	0.000025
CM-Project A	0.044860	-0.000019
CM-Project B	0.014165	0.000222
CM-Project C	0.027830	-0.000001
Average:	0.075647	0.000013
OS Average:	0.122341	-0.000042
CM Average:	0.028952	0.000067

TABLE 6
Functions Modified

Project	Functions Modified As A Percentage of Total Functions, AveragedAcross All Releases
OS-Linux	17.47%
OS-Apache	7.78%
OS-GCC	1.60%
CS-Project A	3.07%
CS-Project B	3.90%
CS-Project C	2.77%
Average - All Projects	6.10%
Average - All OS Projects	8.95%
Average - All CS Projects	3.24%

and fixed faster, we would expect the graphs for changing rate to start off high and to decrease at a sharp slope over time. Therefore, an examination of the linear model parameters can identify these trends, as is shown in Table 5. We can see that the y-intercept for open-source projects is generally higher than the y-intercept for closed-source projects, indicating that more functions were initially changed. In addition, the slope for open-source projects generally has a steeper negative slope, indicating that less and less functions are being modified over time. This supports the hypothesis that more defects are found and, subsequently, the product becomes more stable over time.

The number of modules changed as a percentage of the total number of modules in the release is also examined, and the results averaged across all of the releases. The data from our analysis is shown in Table 6. The open-source projects in our analysis have an average percentage of functions modified of 8.95 percent, significantly larger than the value of 3.24 percent, which is the average percentage of functions modified for closed-source projects. This also supports the hypothesis that more defects are being found and fixed over time in open-source projects than in closed-source projects. The Linux and Apache projects have the largest percentages, followed by the closed-source Project B. Linux shows the largest average of functions modified as a percentage of the total functions, but this is due to the granularity of our sampling. Since our analysis focuses only on the major developmental releases for Linux, we get an artificially high number for functions added and modified for the Linux project.

4.5 Modularity

Gall et al. [3] also compare the growing rate to the changing rate over time for their system. Their results show that the growing rate and the changing rate track each other for some of the releases and, in other releases, the changing rate increases when many new programs were added. The authors find that no significant correlation between the two rates can be established. One of the main differences in our results is that the changing rate is typically lower than the growing rate for the majority of projects analyzed. For the project analyzed by Gall et al. [3], the changing rate was always higher than the growing rate. This may imply that the projects in our analysis were in a more stable phase of development, which resulted in less feature addition and higher maintenance. Another difference was how our results demonstrated correlation. A summary of the correlation calculated between the functions added and the functions modified is shown in Table 7. High positive significant correlation is demonstrated by the open-source projects, yet the closed-source projects do not show any significant positive correlation.

As mentioned in the literature survey, O'Reilly [9] notes that extensibility is a characteristic of open-source systems. If open-source systems are generally more extensible than closed-source systems, then there should be more coupling in closed-source systems. With tighter coupling, adding a feature would require more changes to existing modules to support such a feature. Therefore, we expect that closed-source projects would exhibit stronger positive correlation between functions added and functions changed compared

TABLE 7
Correlation between the Growing Rate and the Changing Rate

Project	Correlation
OS-Linux	-0.03
OS-Apache	0.69
OS-GCC	0.77
CM-Project A	0.65
CM-Project B	0.96
CM-Project C	0.95

to open-source projects. However, our analysis finds the opposite for the projects we investigated. For the open-source projects in our analysis, the number of functions added per release is strongly related to the number of functions modified per release. For the closed-source projects, the number of functions added and modified seems unrelated. This may mean that the development model itself may influence how projects evolve; perhaps not in terms of the overall growth but in how modules are added and modified. Future research may involve validating this finding against other closed-source and open-source projects and attempting to explain why this correlation is as strong as it is in open-source projects.

5 Conclusion

A summary of the results is shown in Table 8. Our analysis does not support the hypothesis that open-source development fosters faster system growth than closed-source development. When compared using a linear model, the slopes of all of the projects in our analysis are similar. This could indicate that other external factors have more impact on the evolutionary characteristics of the software than the

TABLE 8
Hypotheses and Our Results

Current Research:	Metric:	Our Results:
Open source development fosters faster system growth.	Overall project growth in functions over time.	**Does not support hypothesis.** The project growth rates over time were similar for all of the projects in our analysis, open source projects did not exhibit significantly higher growth rates than closed source projects.
	Overall project growth in LOC over time.	
Open source projects foster more creativity.	Functions added over time.	**Supports hypothesis.** The growing rate of the open source projects in our analysis was greater than the closed source projects.
Open source projects succeed because of their simplicity.	Overall project complexity	**Does not support hypothesis.** The closed source projects in our analysis were generally simpler than the open source projects analyzed.
	Average complexity of all of the functions	
	Average complexity of the functions added.	
Open source projects generally have fewer defects than closed source projects, as defects are found and fixed rapidly.	Functions modified over time.	**Supports hypothesis.** The y-intercept of the linear approximation of the functions modified over time was higher in open source projects. The number of functions modified as a percentage of the total functions was generally higher in open source projects.
	Functions modified as a percentage of total functions.	
Open source projects are more modular than closed source projects.	Correlation between functions added and functions modified.	**Does not support hypothesis.** There was strong significant correlation between the growing rate and the changing rate in open source projects, but little to no correlation in closed source projects.

economic model for the production of the software itself. This may also indicate that we have independently validated Lehman's FEAST hypothesis [6] that external global factors and feedback mechanisms have a greater impact on the evolution of software than the development method. If this is the case, then our research implies that whatever these feedback mechanisms or global processes are, they are shared in both open-source and closed-source project development environments. Future research may be able to explore this further, and provide a hypothesis for what these shared processes or feedback mechanisms are.

The hypothesis that open-source software fosters more creativity is supported by our analysis. The growing rate, or the number of functions added, was greater in the open-source projects than in the closed-source projects. This indicates that the open-source approach may be able to provide more features over time than by using the closed-source approach. Practitioners interested in capturing market share by providing additional features should look to the open-source methodology as a method to achieve this.

In terms of defects, our analysis finds that the changing rate or the functions modified as a percentage of the total functions is higher in open-source projects than in closed-source projects. This supports the hypothesis that defects may be found and fixed more quickly in open-source projects than in closed-source projects and may be an added benefit for using the open-source development model. As was noted by Wheeler [13], open-source software projects often have a large number of testers since a large portion of the users become the testers. The users are also able to identify the defects and may even provide a solution.

Our analysis did not support the hypothesis that open-source projects are more modular. The correlation between the functions added and the functions modified is significant in open-source projects, suggesting that adding features in open-source projects requires additional changes to existing modules because of tight coupling. In closed-source projects, there was little to no correlation. This indicates that open-source projects may be prone to problems with coupling. Open source system architects should be mindful of this possibility and implement strategies to reduce coupling when using open-source development methods.

The hypothesis that open-source projects are generally less complex than closed-source projects is not supported by our analysis. All of our metrics in our analysis show that the closed-source projects are less complex than open-source projects in our analysis. Practitioners who are starting an open-source project may wish to implement coding practices and guidelines that limit individual function complexity. This may also reduce the tight coupling as was evidenced by the correlation between the functions added and the functions modified.

In summary, our research validates two of the key perceptions about open-source software, while three common perceptions were not supported by our analysis. Companies considering the move to using an open-source development method for future projects should consider all of the goals they are trying to achieve and ensure that proper metrics are collected and monitored to ensure these goals are achieved. Future research into this area may further validate these perceptions, and expand the empirical data available for review.

ACKNOWLEDGMENTS

The authors would like to thank the University of Calgary, the Free University of Bozen, NSERC, and ASERC for partially supporting this research.

REFERENCES

[1] C. Cook and A. Roesch, "Real-Time Software Metrics," *J. Systems and Software,* vol. 24, pp. 223-237, 1994.
[2] J. Dalle and N. Jullien, "Windows vs. Linux: Some Explorations into the Economics of Free Software," *Proc. Acts of SSII Conf.*, 2000.
[3] G. Gall, M. Jazayeri, R.R. Klosch, and G. Trausmuth, "Software Evolution Observations Based on Product Release History," *Proc. Int'l Conf. Software Maintenance,* 1997.
[4] M.W. Godfrey and Q. Tu, "Evolution in Open Source Software: A Case Study," *Proc. Int'l Conf. Software Metrics,* 2000.
[5] C.F. Kemerer and S. Slaughter, "An Empirical Approach to Studying Software Evolution," *IEEE Trans. Software Eng.*, vol. 25, no. 4, pp. 493-509, July/Aug. 1999.
[6] M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, and W.M. Turski, "Metrics and Laws of Software Evolution—The Nineties View," *Proc. Fourth Int'l Metrics Symp.,* 1997.
[7] A. Mockus, R.T. Fielding, and J. Herbsleb, "A Case Study of Open Source Software Development: The Apache Server," *Proc. 22nd Int'l Conf. Software Eng.*, 2000.
[8] Mozilla, "Our Mission," http://www.mozilla.org/mission.html, Aug. 2003.
[9] T. O'Reilly, "Lessons From Open-Source Software Development," *Comm. ACM,* vol. 42, no. 4, pp. 32-37, 1999.
[10] J.W. Paulson, "An Empirical Study on the Growth of Open Source and Commercial Software Products," master's thesis, Dept. of Electrical and Computer Eng., Univ. of Calgary, Alberta, 2001.
[11] K. Pearson, "Mathematical Contributions to the Theory of Evolution. III. Regression, Heredity and Panmixia," *Philosophical Trans. Royal Soc. of London,* 1896.
[12] E.S. Raymond, "The Cathedral and the Bazaar," http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/, Aug. 2003.
[13] S. Wheeler, "Open Source Software," http://sucs.org/sits/articles/opensource/opensource.pdf, Aug. 2003.
[14] D. Wheeler, "Why Open Source Software/Free Software (OSS/FS)? Look at the Numbers!," http://www.dwheeler.com/oss_fs_why.html, Aug. 2003.

James W. Paulson received the BSc degree in electrical engineering from the University of Saskatchewan in 1993, and the MSc degree in software engineering from the University of Calgary in 2002. He is currently employed with General Dynamics Canada, located in Calgary, Alberta. He is a senior software engineer and is involved in developing communications software for military applications. He has held several positions in the software engineering industry over the past 10 years, developing both embedded and application software. He is a member of IEEE and is a registered professional engineer in the province of Alberta.

Giancarlo Succi received the Laurea degree in electrical engineering (Genova, 1988), the MSc degree in computer science (SUNY Buffalo, 1991), and the PhD degree in computer and electrical engineering (Genova, 1993). He is a tenured professor at the Free University of Bolzano-Bozen, Italy, where he directs the Center for Applied Software Engineering. Before joining the Free University of Bolzano-Bozen, he has been a professor at the University of Alberta, Edmonton, an associate professor at the University of Calgary, Alberta, and an assistant professor at the University of Trento, Italy. He was also chairman of a small software company, EuTec. He has been a registered professional engineer in Italy since 1991 and he obtained the full registration also for the province of Alberta, Canada, while residing there. His research interests involve multiple areas of software engineering, including 1) open-source development, 2) agile methodologies, 3) experimental software engineering, and 4) software product lines and software reuse. He has written more than 150 papers published in international journals, books, and conferences, and is the editor of four books. He has been a principal investigator for projects amounting more than 5 million dollars in cash and, overall, he has received more than 10 million dollars in research support from private and public granting bodies. He has chaired and cochaired several international conferences and workshop, is a member of the editorial board of international journals, and a leader of international research networks. He is a consultant for several private and public organizations worldwide in the area of software system architecting, design, and development; strategy for software organizations; training of software personnels. Dr. Succi is a Fulbright Scholar and a member of the IEEE Computer Society.

Armin Eberlein received the Dipl-Ing (FH) degree in telecommunications engineering at the Mannheim University of Applied Sciences in Germany, the MSc degree in communications systems, and the PhD degree in software engineering from the University of Wales, Swansea, UK. He is an associate professor and is currently on leave from the Department of Electrical and Computer Engineering at the University of Calgary, Canada, where he served as a codirector of the Alberta Software Engineering Research Consortium (ASERC) and as the director of the Software Engineering Program. He spends his leave in the Computer Engineering Department of the American University of Sharjah in the United Arab Emirates. His research interests focus on the improvement of requirements engineering practices and techniques. He has worked previously as a hardware and software developer at Siemens in Munich, Germany, and has consulted for various companies in Germany, the UK, and Canada. He is a member of the IEEE Computer Society.