Exploring Complexity in Open Source Software: Evolutionary Patterns, Antecedents, and Outcomes

David P. Darcy
Irish Management Institute
david.darcy@imi.ie

Sherae L. Daniel
University of Pittsburgh
sldaniel@katz.pitt.edu

Katherine J. Stewart
University of Maryland
kstewart@rhsmith.umd.edu

Abstract

Software complexity is important to researchers and managers, yet much is unknown about how complexity evolves over the life of a software application and whether different dimensions of software complexity may exhibit similar or different evolutionary patterns. Using cross-sectional and longitudinal data on a sample of 108 open source projects, this research investigated how the complexity of open source project releases varied throughout the life of the project. Functional data analysis was applied to the release histories of the projects and recurring evolutionary patterns were derived. There were projects that saw little evolution, according to their measures of size and structural complexity. However, projects that displayed some evolution often differed on the pattern of evolution depending on whether size or structural complexity was examined. Factors that contribute to and result from the patterns of complexity were evaluated, and implications for research and practice are presented.

1. Introduction

Software complexity plays a pivotal role in software engineering practice and research [1, 2]. Software complexity is an outcome of problem selection, design choices, and implementation details, and an input to various long-term consequences such as quality and maintainability [3-6]. For example, high levels of complexity have been used to identify the parts of a software application that are most prone to faults and thus direct where developer effort could be best focused [7]. Thus managing complexity is of central importance to project success, and the topic of software complexity is of critical interest to developers and researchers.

This paper seeks to add to the literature by using a new statistical method to identify recurring patterns of complexity evolution in open source software (OSS) development projects and exploring potential causes of and effects from a project following a particular evolutionary pattern.

While much work to-date has examined software complexity in the context of closed source software development [8, 9], some unique characteristics of OSS development may increase the importance of complexity in the OSS context. OSS is software distributed under a license approved by the Open Source Initiative (OSI). For a license to be approved by OSI it must include many provisions, one of the most prominent being that source code must be available to the user (see http://www.opensource.org/docs/definition.php).

Given its wide availability, OSS is often developed by volunteer workers. When developers may come and go without formal integration of new contributors or turnover processes when a contributor leaves, high complexity may severely inhibit new members' ability to understand the code and contribute effectively. Difficulty in maintaining continuity as turnover occurs may be ameliorated (or exacerbated) by the presence (or absence) of documentation [10]. However, while research on software documentation finds that the availability of documentation increases the quality of the code design for complex tasks [10], formal documentation of requirements or design are often lacking in OSS projects [11].

Determining what associations exist between starting and ending conditions and the evolutionary path followed by a project may yield useful insights regarding how project administrators and managers might steer projects towards successful ends. For example, managers may be able to use such findings to start projects in a way that the complexity is likely to be maintained at a low level. Further, by identifying evolutionary paths, OSS managers could identify which path their application is following, and if it is a path leading to undesirable outcomes, they can make changes.

This research has three goals. The first is to identify patterns of evolution in OSS projects by examining two important dimensions of the software code, size and structural complexity, and how they change over the first three years of a projects' public life. The second goal is to explore whether the

IEEE computer society

978-0-7695-3869-3/10 $26.00

evolutionary patterns followed are associated with certain starting conditions, in particular initial size, initial structural complexity, or initial levels of internal documentation. The third goal focuses on determining whether evolutionary patterns are associated with different outcomes in size, structural complexity, or internal documentation.

The next section provides a brief discussion of software complexity, and reviews related prior research. Sections 3 and 4 contain a description of the research design and functional data analysis. Section 5 describes the patterns identified and explores their antecedents and outcomes. Conclusions, limitations, and future research are discussed in Section 6.

2. Previous Research

Software complexity impacts the degree of difficulty a developer has in understanding the software, and it may vary greatly across different software implementations developed to address the same underlying problem [12]. Over the life of a project it is often necessary to alter software so that it continues to meet changing user needs, and with alteration, software complexity evolves. When developers add features to an application, it is common that each additional feature will add complexity to the software [2]. However, it is important to minimize this effect in order to ease the difficulty of adding features or fixing bugs in the future [12].

2.1. Dimensions of Software Complexity

Size is an oft cited dimension of software complexity, typically measured as source lines of code (SLOC). Software with more SLOC has higher complexity, a higher number of errors, and more developers are needed to complete the tasks related to the software [13-15]. Size is a measure with high accessibility to managers, although other measures that capture complexity irrespective of size may be more helpful for focusing maintenance efforts within a project [16].

Another aspect of software complexity is structural complexity, which captures the way that code is organized rather than how much code exists. Structural complexity is viewed as "the organization of program elements within a program" [17]. An element can refer to a procedure, function, method, module, class, etc. When solving a problem through software, several elements are typically created. As design, implementation and maintenance decisions are made, the content of these elements and the relationships between these elements leads to the structural complexity of the software. That is, the structural complexity of a program depends on the complexity of individual elements and the complexity of the associations among these elements. The structure of elements created by the developers can have several implications including affecting the ease with which the code can be altered as bugs are fixed and features are added, and the reliability of the application.

Structural complexity is reflected in both cohesion and coupling. Cohesion is conceptualized as the 'togetherness' of the sub-elements within an element [18]. Higher cohesion indicates lower complexity. Coupling focuses on the associations among elements, and is conceptualized as the 'relatedness' of elements to other elements [19]. As coupling increases, so does complexity.

Coupling and cohesion have been found to interdependently affect effort, an outcome of substantial managerial interest [12]. Based on such findings, it can be argued that a single measure of structural complexity for a project may be calculated using coupling and cohesion, and such a measure adequately captures overall structural complexity [12].

2.2. Software Complexity Evolution

Size and structural complexity may evolve in different ways. Manduchi and Taliercio studied the evolution of complexity in a closed source setting over the course of three software releases [20]. They observed multiple dimensions of software complexity including the complete set of Chidamber and Kemerer metrics and size, and they found that all measures increased, but at different rates. Similarly, Yu, Schach, Chen and Offutt, examined the evolution of 400 successive versions of the Linux kernel and found that both size and common coupling with non-kernel modules increased, but also at different rates [9]. Specifically they found that size increased in a linear fashion, while common coupling increased at an exponential rate.

In contrast to these studies in which all measures increased over time, MacCormack, Rusnak and Baldwin found that a redesign of Mozilla resulted in reductions in complexity as indicated by both coupling and size [21]. After redesign, analysis of further evolution indicated that coupling was maintained at a low level while size grew. This result, where one measure of complexity increases and another decreases, is especially striking when considering Lehman's law suggesting that unless attended to, complexity will increase over time [22]. An interesting question is whether this project is an anomaly or whether many projects display increases in size while also lowering their level of complexity based on other measures. This question gains importance if one considers increases in size a necessary way to add functionality. It is important to understand if functionality can continue to be added while also

maintaining a minimal level of structural complexity, and what may facilitate a project following such a preferred evolutionary path.

2.3. Summary

Software complexity is both an indicator of the success of an OSS project as well as a likely contributor to other important outcomes including the ability of the project to attract community input and market share [23]. Empirically studying complexity may be useful to help understand how complexity may be managed in OSS development to produce software that is more easily maintained, and how complexity can be used to evaluate the expected viability of an OSS project. In particular, size and structural complexity are two dimensions of software complexity that have been well validated as predictors of external attributes such as effort, productivity and rework [2]. However, these measures are distinct and there is evidence that size and other measures of complexity follow distinct evolutionary patterns. We seek to identify patterns of evolution for size and structural complexity and investigate their antecedents and consequences.

3. Research Design

Research was conducted in four stages. In the first stage, sampling criteria were defined, projects to include were identified, source code from all available releases of those projects was downloaded, the data was cleaned, and measures of interest were calculated for every release of every project. In the second stage functional data analysis was applied to uncover patterns of evolution in size and in structural complexity and to categorize projects accordingly. This analysis produced tentative conclusions as to how complexity evolves in OSS projects. To expand and strengthen those conclusions, the third stage of the study explored whether starting conditions could be identified that were associated with the evolutionary pattern complexity followed and whether there were significant differences in ending conditions based on the pattern followed. These analyses were conducted for both size and structural complexity. The final stage of the analysis compared the categorization of projects using size and structural complexity measures.

3.1. Data Site

The sample of projects was drawn from SourceForge (www.sourceforge.net). SourceForge provides open source developers a centralized place to manage development and includes communication tools, version control processes, and repositories for source code.

Drawing a sample from this site allows this study to build on prior open source research by focusing on a larger and more diverse set of projects compared to previous studies of individual projects such as Linux, Mozilla or Apache. Also, by using SourceForge rather than a single firm, the analysis will add more general results to prior literature on software development that may be unique to a particular firm's idiosyncrasies.

3.2. Project Selection

SourceForge hosts many different kinds of software projects, utilizing many different programming languages. To limit variability due to the nature of the underlying problem addressed across projects, projects were selected from two of the largest problem domains listed on SourceForge, Internet and Networking. To avoid variability due to the project programming language, projects were selected that were built with C++. Within the set of C++ projects in the Internet or Networking domains, those that appeared to be a sub-project of a larger project or that appeared to be an umbrella project for multiple smaller efforts were eliminated. These projects were eliminated because they represented multiple parallel development streams that might not be able to be disentangled, and because some variables were measured at the level of the SourceForge project, and those measures could not be assigned to a single subproject. Finally, to be included in the sample, a project had to have posted at least one release of software code. Applying the selection criteria generated a total of 108 projects for analysis.

3.3. Measurement

Scientific Toolwork's Understand (version 1.4) was used to analyze source code and capture measures for each release. Size was calculated by summing the total number of SLOC in the release. Similarly, the total number of lines of comments in the release was calculated and divided by the SLOC in order to calculate a measure of internal documentation, the ratio of comments to code. Coupling and cohesion were individually assessed for each class in a particular release of the project. To calculate a release level coupling measure, the class level coupling measures were averaged across all classes for the given release of the project. The same procedure is used to assess the release level measure of cohesion. As increasing cohesion represents decreasing complexity, a reversed measure was used, the lack of cohesion, so that increases in the lack of cohesion measure represent increasing complexity. Following [12], to calculate a single measure of structural complexity, the release level measures of coupling and lack of cohesion were multiplied to produce a term referred to below as CplXLCoh. The higher the value of this term for a

given software release, the higher the structural complexity of that release [12].

In addition to the measures of complexity and internal documentation, basic descriptive statistics were calculated. These were the total number of releases for a project, the average number of days between releases (release frequency), and the active life. The active life for a project represents time in number of days between the first and the last release by a project.

3.4. Sample Descriptive Statistics

Descriptive statistics are in Table 1. The average active life across the 108 projects is 271 days. Projects with only one release or with multiple releases all on a single day had an active life of 0. On average, the projects released 7 versions and released them every 39 days. On average, the first release contained just under four thousand SLOC and the final release contained just over six thousand SLOC.

Table 1: Project Descriptive Statistics (n = 108)

	Mean	St. Dev	Min	Max
Active life	271.32	341.68	0	1095
No. of releases	7.06	9.76	1	76
Average release frequency	39.31	53.69	0.00	346.67
SLOC first release	3912	4943	99	24218
SLOC last release	6134	8459	99	48361
CplXLCoh first release	1.52	1.35	0.00	7.20
CplXLCoh last release	1.59	1.26	0.00	5.81
% change SLOC first to last	0.76	2.25	-0.70	17.13
% change CplXLCoh first to last	0.18	0.60	-0.85	3.15

4. Analysis

Functional Data Analysis (FDA) enables an efficient representation of many types of data series in a compact and analytically powerful form (see http://functionaldata.org/ or [27]). FDA enables the creation of functional forms that represent the complexity evolution pattern for a single OSS project, and these are the unit of analysis for this study. A functional form can be analyzed using cluster analysis.

Each functional form is created using two different types of inputs, parameters and observations. The first set of inputs consists of three main parameters that are required for the creation of all functional forms and dictate how the release data for a single project is summarized in a functional form. These parameters are the same for all functional forms in this study. The first parameter is the order of the basis functions. Each functional form is composed of a linear combination of polynomial segments, referred to as basis functions, and the order determines the flexibility of these basis functions. Standard FDA practice of using two for the order of the basis functions is adopted [25]. The second parameter determines the number and spacing of the knots, where the basis functions are joined. The number of knots was specified to reflect the average release frequency of the projects (every 39 days). The final parameter is the smoothing parameter, and it specifies the faithfulness of the basis functions to the supplied data points. A relatively high smoothing parameter of 100 was selected to reduce the jaggedness of the functional form and enable a simpler comparison across projects. Sensitivity analysis in prior work showed that modification of these parameters did not have a substantial effect on conclusions drawn from analysis [26].

4.1. Creating Functional Forms from Release Data

OSS projects are not easily compared due to the many differences across projects and within the same project over time. Three sample projects are depicted in Figure 1 and discussed below to explicate these differences and demonstrate the steps taken to overcome them in order to create comparable functional forms for the OSS projects. In Figure 1, each project's releases are represented by a different symbol.

There are several differences across projects that represent challenges to analysis. One difference across the projects is the time of initial and final release. Notice the first release for the project represented by triangles was made on May 1, 2001, while the first release for the project represented by the circles is December 4, 2002, and there are similar differences in the dates of the last releases for each project. A second difference across projects is the time over which they post releases. The project represented by triangles releases over a distinct period compared to the project represented by the circles. A third difference is the number of releases across projects. The project represented by squares has 16 releases while the project represented by triangles has 6 releases.

One final challenge is that the releases for an individual project may span a different absolute level of complexity compared to the other projects. For example, the project represented by circles has structural complexity varying from 0-2, while the project indicated by squares has releases with structural complexity between 2 and 5.

To address these challenges a conversion process was applied to each project release history before the histories were converted into functional forms. In FDA

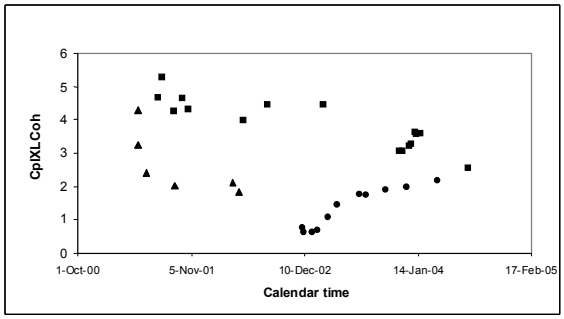

Figure 1: Three project release histories

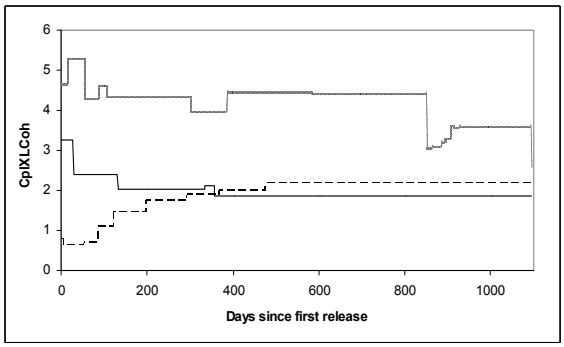

Figure 2: Three projects, feature aligned

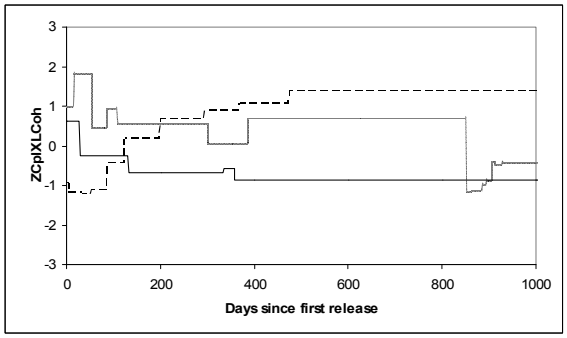

Figure 3: Three projects, standardized

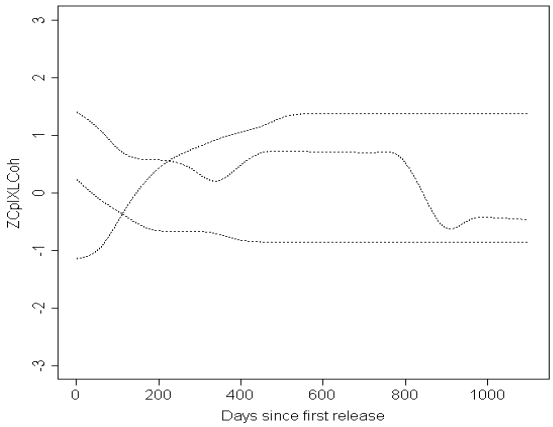

Figure 4: Three projects, functional objects

parlance, this conversion is known as feature alignment. To address the first challenge related to staggered starting points, project histories were aligned according to their first release of code on SourceForge. The x-axis then becomes days since the first release rather than actual calendar time, as depicted in Figure 1. To address the challenge related to staggered end points, a 3-year history (1095 days since the first release) was chosen as being sufficiently long for projects to display considerable evolution, and likely long enough to capture the active life of most projects [26]. For projects that did not have a release on the final day, the value of complexity for their last release before day 1095 is used until day 1095. This same process was used to impute values for complexity on all days between releases. In other words, the days between releases were assigned the value of complexity from the most recent release. The same three projects from Figure 1 are represented in Figure 2 after these adjustments.

Finally, in order to focus on evolutionary patterns rather than absolute differences, the vertical shift across projects is minimized by standardizing the complexity values within each project. This focuses attention on within project evolutionary dynamics. Data following standardization are shown in Figure 3.

Using the two sets of input, the converted project release data and the FDA parameters, a functional form was generated to represent the evolution of complexity for each project, for size and structural complexity. Resulting functional forms for the three sample projects are displayed in Figure 4.

The data processing and analysis steps described above were conducted once to create functional forms for size and once for structural complexity. Results for size are discussed first, then for structural complexity, then a comparison across the two.

The functional forms representing the evolution of SLOC for all projects are shown in Figure 5. Figure 5 also includes a functional form in bold that represents the average of all the projects and the 95% confidence interval around that average is indicated by bold dashed lines. The individual projects are difficult to distinguish and the mean curve and confidence interval are fairly flat, trending slightly upwards at first and flattening towards the end of the period of study.

5. Results

Having created functional forms for every project for both size and structural complexity, the analysis proceeded to uncover patterns in evolution using these two measures and investigating possible contributing factors to and impacts of those patterns. To identify patterns, cluster analysis is used. After creating clusters, the General Linear Model procedure (GLM)

was used to explore contributing factors and impacts. These results are reported separately for analyses of size and structural complexity. Differences in project classification across the size and structural complexity analyses are explored in the last subsection.

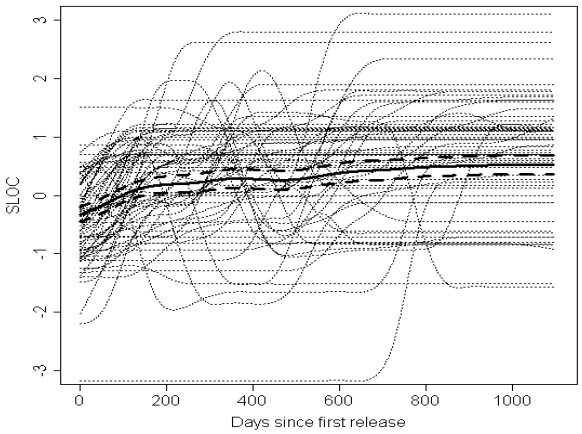

Figure 5: All projects, functional objects (n = 108)

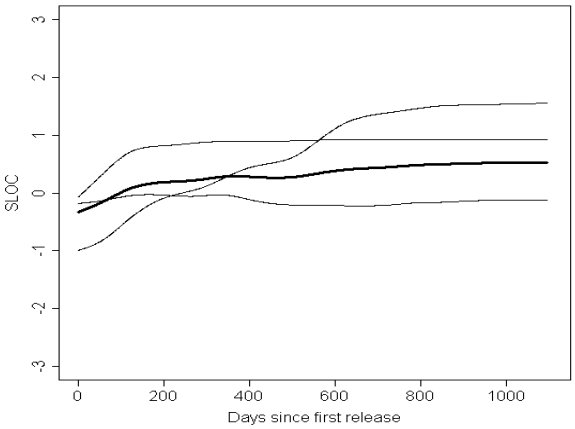

Figure 6: Mean size curve for all projects (bold) and three cluster mean curves

5.1. Patterns of Evolution in Size

Two methods are used to uncover patterns in FDA: principal components analysis and cluster analysis [27]. Principal components analysis is more useful to detect variations from a single overall pattern whereas cluster analysis is better for detecting different overall patterns. Thus cluster analysis is used here. K-medoids clustering is used to reduce the impact of outliers in shaping the clusters [27]. Because there is no theoretical reason to expect a particular number of clusters, the two-cluster solution was examined first then clusters were added until no new insight was gained by adding an additional cluster. A three cluster solution yielded the most explanatory power and provided the most interpretable results for the evolution of SLOC. This solution is shown in Figure 6.

The bold curve in Figure 6 represents the overall mean and each of the other lines represents a cluster mean. The largest cluster of projects (n = 55) is represented by the relatively flat line; on average, these projects had very little change in SLOC over the observed 3-year history. Below, projects in this cluster are referred to as the "no growth" projects. The next largest cluster (n = 29) contained projects that, on average, had a relatively sharp increase in SLOC for approximately the first 200 days of their public development and then showed essentially no change for the remainder of the 3 years. Below, this cluster is referred to as the "short growth" projects. The last cluster (n = 24) contains projects that, on average, showed a steady increase in SLOC for approximately the first 600 days, with a much slower rate of increase after that, for the remainder of the 3 years. Below, these are referred to as the "long growth" projects.

The clusters are visually distinguished mainly by the amount and period of growth. To confirm that the clustering resulted in groups that were significantly different from one another, the General Linear Model (GLM) procedure in SPSS 14.0 was used to assess the differences in the percentage change in SLOC and the active life across clusters. Both effects were significant at p <= .001 (R-square 17.6% for the percentage change in SLOC, R-Square 31.0% for active life) and differences across clusters were in the expected directions, supporting the proposition that the clustering did result in the separation of projects into distinct clusters that were significantly different in the growth patterns displayed.

The within cluster change in SLOC was also examined to confirm that changes in SLOC implied by the appearance of the average cluster line were of the relative magnitude and significance implied by the picture in figure 6. All clusters showed average increases in SLOC from the first to the last release. For the short growth projects the difference was 1536 SLOC (p = .009), and for the long growth projects the difference was 7594 SLOC (p = .000). The no growth projects showed a much smaller and only marginally significant change (difference = 238 SLOC, p = .066).

5.2. Size Patterns: Antecedents and Consequences

The next step in the analysis was to explore whether there were some starting conditions that are

associated with the SLOC evolutionary path. The level of complexity in the first release of the software was examined including the measures of SLOC, CplXLCoh, and individual measures of coupling and cohesion. The internal documentation in the first release was also considered, with the expectation that projects with a higher initial ratio of comments to code should be easier to build on and would thus lead to more OSS development. For this analysis, the GLM procedure was again employed, with cluster membership (whether a project was in the no growth, short growth, or long growth cluster) as the between projects factor. Given the exploratory nature of the work, p = .10 was used as the cutoff for interpreting effects. The only variable that had a significant multivariate effect was initial coupling (F = 3.128, p = .048). Mean contrasts showed that initial coupling was higher for both the short growth (mean = 3.06, p = .058) and long growth (mean = 3.19, p = .035) clusters than for the no growth cluster (mean = 2.31), but not significantly different between the short and long growth clusters (p = .760). In other words, projects that started with a higher level of coupling in their first release were more likely to grow than those that started with a lower level.

Finally, significant differences that existed in SLOC, CplXLCoh, individual measures of coupling and cohesion, and internal documentation at the end of the observed period were examined. The GLM results indicated there was a significant difference in ending SLOC (p = .000) and in the ending value of coupling (p = .010). Specifically, projects in the long growth cluster ended up significantly larger than the no growth projects (p = .000) and than the short growth projects (p = .001). The pattern of differences in coupling was the same at the end as it had been at the start of the projects.

5.3. Patterns of Evolution in Structural Complexity

As with SLOC, a 3-cluster solution produced the most interpretable results for the analysis of structural complexity. Figure 7 depicts the evolution in ZCplXLCoh across the 3 clusters. The largest cluster of projects (n = 53) is represented by the relatively flat line; on average, these projects had very little change in ZCplXLCoh over the 3-year history. Below these are referred to as the "unchanging structural complexity" projects. Projects in the next largest cluster (n = 38) had projects an increase in ZCplXLCoh for about the first 600 days of their public development and then showed essentially no change for the remainder of the 3 years. Below these are referred to as the "increasing structural complexity" projects. The last cluster (n = 17) contains projects that, on average, showed a decrease in ZCplXLCoh for approximately the first 600 days. Below these are referred to as the "decreasing structural complexity" projects.

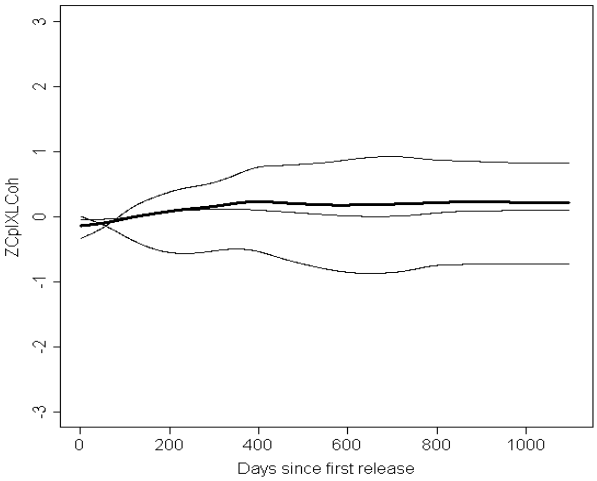

Figure 6: Mean complexity curve for all projects (bold) and three cluster means

The same series of analyses was applied to this clustering as was applied to the SLOC clusters. The GLM procedure was used to assess the differences in the percentage change in ZCplXLCoh and the active life across clusters. Both effects were significant at p <= .01 and differences across clusters were in the expected directions, supporting the conclusion that the clustering did result in groups of projects that were significantly different in the patterns of change displayed. The within cluster change in ZCplXLCoh was also examined to confirm the patterns observed. Results were as expected. For the increasing structural complexity projects the difference between starting and ending ZCplXLCoh was 0.323 (p = .005). For the decreasing structural complexity projects the difference was -0.388 (p = .022). The unchanging structural complexity projects showed no significant difference (0.038, p = .740).

5.4. Structural Complexity Patterns: Antecedents and Consequences

As with the analysis of SLOC, it was explored whether the starting conditions of a project were associated with which evolutionary path it followed for ZCplXLCoh. There were significant multivariate effects for internal documentation (F = 3.032, p = .053), initial CplXLCoh (F = 3.018, p = .053), and initial coupling (F = 2.924, p = .058). Comparisons across clusters demonstrated that the internal documentation was higher for both of the evolving

clusters (increasing and decreasing ZCplXLCoh) than for the unchanging cluster (difference = 0.297, p = .021). Initial CplXLCoh was lower for the increasing CplXLCoh than the decreasing CplXLCoh cluster (difference = .916, p = .020) and higher for the decreasing than the unchanging cluster (difference = .796, p = .033). Coupling was higher for the decreasing than the increasing CplXLCoh cluster (difference = .977, p = .061), lower for the unchanging than the combined evolving clusters (difference = 1.379, p = .055), and higher for the decreasing CplXLCoh cluster than the unchanging cluster (difference = 1.178, p = .018).

Finally, significant differences that existed in the level of SLOC, CplXLCoh, or internal documentation at the end of the observed period were explored. The GLM results indicated there was a significant difference in ending SLOC (F = 5.676, p = .005), internal documentation (F = 3.584, p = .031) and in the ending value of coupling (F = 2.419, p = .094). Specifically, the increasing CplXLCoh projects ended up with higher SLOC (difference = 6314.294, p = .010) than the unchanging projects (difference = 5452.882, p = .003). The increasing projects ended less well documented than the decreasing projects (difference = .169, p = .096), and the combined evolving clusters were better documented than the unchanging projects (difference = .340, p = .016). The decreasing projects were better documented than the unchanging projects (difference = .254, p = .009).

5.5. Comparing Size and Structural Complexity Project Classifications

Table II shows the assignment of projects to SLOC clusters and to ZCplXLCoh clusters. Over all projects, there was a significant positive correlation between the percent change in SLOC and the percent change in ZCplXLCoh (r = 0.221, p = .024). However, analyzing each of the ZCplXLCoh clusters separately showed that this correlation was not significant for the increasing structural complexity projects (r = 0.174, p = 0.31), was positive and significant for the unchanging structural complexity cluster (r = 0.275, p = 0.05), but negative and significant for the decreasing structural complexity cluster (r = -0.52, p = 0.03).

Table 2: Cluster cross-membership

		SLOC			
		Long growth	Short growth	No growth	Total
ZCplXLCoh	Increasing	13	19	6	38
	Unchanging	8	2	43	53
	Decreasing	3	8	6	17
Total		24	29	55	108

6. Discussion

The main objectives of this study were: the identification of different patterns of evolution in OSS projects and an assessment of contributing factors and consequences for different patterns of evolution. This section discusses conclusions and implications. Limitations are summarized prior to the conclusion.

6.1. Evolutionary Patterns Uncovered

Analyses of size and structural complexity identified sets of projects that displayed very little evolution. This set is referred to as the no growth cluster for SLOC and the unchanging structural complexity cluster for ZCplXLCoh. There was significant overlap in the categorization of individual projects into these clusters, with 78% of projects categorized as no growth based on SLOC also categorized as unchanging based on ZCplXLCoh (see TABLE II). Overall, 39.8% of the projects were categorized this way. However, for the remaining 60.2% of the projects, there was no clear correlation between their categorization based on an analysis of size and their categorization based on analysis of structural complexity.

Analysis of SLOC is consistent with expectations based on much prior work: all evolving projects grow, though to different extents and over different time periods. Analysis of ZCplXLCoh, however, provides some more surprising insight. By analyzing ZCplXLCoh, a set of projects was identified that follow a decreasing pattern over time. Comparison across the two sets of cluster analyses showed that these projects would not be identifiable by their pattern of change in SLOC. They fell into all three of the SLOC clusters and demonstrated a significant negative correlation between growth in SLOC and change in ZCplXLCoh. Identification of this cluster of projects provides some evidence that there are a group of projects that stand in contrast to Lehman's second law: they grew in size (on average 17%), while decreasing their structural complexity (on average 13%). Thus in exploring potential antecedents to and outcomes from the evolutionary patterns, this research was particularly interested in determining what factors may be associated with projects' categorization into the decreasing versus the increasing structural complexity clusters.

6.2. Initial Software Characteristics and Patterns

Assuming that evolution is positive, because it indicates addition of functionality, modification to meet changing user needs, or other enhancements in quality, then there is value in identifying the initial

characteristics of OSS that are associated with its later evolution, or lack thereof.

For both size and structural complexity, projects in the evolving clusters had higher initial levels of coupling. Coupling indicates the extent to which the classes are inter-related. There may be several reasons a high initial level of coupling could spur growth in size. First, if coupling is high, any modification to the code may require changes (i.e., added SLOC) in several related classes/methods. Second, because the classes are highly interrelated, if a person tries to work on one class, the person may have to look at several other classes, which may increase the chance that the person will add more code to those classes. On the other hand, rather than examining and modifying related classes, a developer may simply avoid altering existing classes to add functionality and instead write new code from scratch, reducing re-use and increasing the overall size of the project. Alternatively, a higher level of coupling could indicate that the code has undergone significantly more development prior to initial release than software with lower coupling. If this is true, the code may have greater functionality and thereby attract more developers, spurring growth.

While the first three possibilities specifically address the association between initial coupling and evolution based on size, the latter explanation may also explain why higher initial coupling is associated with evolution as indicated by structural complexity. In analyzing starting conditions associated with evolution in structural complexity, we found that projects in the evolving clusters also had higher initial levels of internal documentation. The most obvious explanation for this association is that projects that are better documented are easier for developers to modify. The fact that this result held for evolution in structural complexity but not size may be because, for projects that began with poor structural complexity and were well documented, efforts were aimed at simplification of the code. Such simplification efforts may be associated with small or even negative changes in size. Indeed, of 17 projects in the decreasing structural complexity cluster, only 3 were also in the long growth cluster.

To summarize, one explanation for the results is that projects beginning with a high level of coupling attract more effort. This effort may result in growth in size, or, for projects that are well documented, the effort may be directed at 'cleaning up' the code, resulting in decreasing structural complexity.

6.3. Ending Software Characteristics and Patterns

Projects that decreased in structural complexity were those that started with higher levels of structural complexity. As noted above, this higher overall level of structural complexity seems to have been driven by a higher overall initial level of coupling. Analysis of the end states of the projects across clusters showed that the final levels of structural complexity were not significantly different. This suggests that projects evolved toward a common level of structural complexity.

To summarize, across structural complexity clusters, all projects started at approximately the same size. What may have determined whether they evolved was the level of internal documentation. For those that were well documented, the starting level of structural complexity determined the trajectory of their evolution. Those that started with high structural complexity exhibited a small growth in size and a significant decrease in structural complexity. Those that started with a relatively low level of structural complexity exhibited some increase in structural complexity, but this accompanied a relatively large increase in size. If size is taken as a proxy for the amount of functionality and structural complexity as a proxy for design quality, projects that began with good design and internal documentation experienced the biggest boost in added functionality. Those that began with good internal documentation but poor design, experienced a small increase in functionality and a significant increase in design quality.

The analysis of size yielded few new insights. Based on the analysis of size, one would conclude that this study's findings support prior work, which has shown systems that evolve become more complex (i.e., bigger) over time. These results add to prior work by identifying two distinct patterns of increase as well as the third pattern of no evolution described above. Though they started without significant differences in size, the long growth projects ended larger than the short growth projects. It may be useful for future research to explore which factors put a project in the long growth versus the short growth trajectory.

6.4. Implications for research

The results have several implications for future research on software evolution. One important finding is that size and structural complexity are not interchangeable indicators of complexity. Much past work has used size as a proxy for software complexity, but findings of this study show that projects whose size follows one pattern of evolution may evidence a very different pattern when structural complexity is examined. Focus on size yields results consistent with Lehman's second law, however the analysis of structural complexity indicates that for a significant set of OSS projects, that law may not hold.

The findings show that in the projects studied, structural complexity tended toward a mean value over time. That is, the cluster of projects that decreased in structural complexity and the cluster that increased both ended up closer to a mean value than when they started. This prompts the question of whether there is a "natural" level of structural complexity that projects achieve over time or whether there may be some factors, unmeasured in this research, that lead to decreasing versus increasing trajectories.

An important question not directly considered in this study is to what extent the OSS milieu contributes to the different evolutionary patterns, particularly when compared to closed source development. Though it would be difficult to access a similar data set of closed source projects, exploring the impact of social organizing principles may be useful.

In addition to the theoretical questions raised by the research, a methodological contribution is presented in the utilization of FDA. The study demonstrates that this is a useful technique for studying evolution across a relatively diverse set of projects following different release schedules. Also, the results here imply that for the study of these kinds of OSS projects, approximately three years of observation may capture most evolution of most projects in that the overall average active life of projects in the sample was 271 days with a standard deviation of 341 days.

6.5. Implications for practice

The main take-away for OSS projects administrators is that better documented code facilitates interest and contributions to projects, thereby generating evolution. This holds even if the initial design of the software is relatively poor. Thus this research indicates that if continuing development effort is a goal, then prior to releasing an initial version time should be invested in thorough documenting, while time spent improving the structure may be less influential.

Specifically, regardless of the trajectory followed, OSS development efforts appeared to be "attracted" to well documented code. The importance of providing good documentation for OSS projects, though often a notoriously low priority activity [28], is underscored by this result. Admonitions to provide good internal documentation in order to encourage more and better development may focus the attention of project administrators and contributors. Such prescriptions may also have value in closed source contexts as well, particularly where different groups execute development and maintenance activities. Further research might also investigate more carefully what kinds of internal documentation are more or less useful in encouraging evolution.

While the OSS community has a reputation for high quality development, ultimate success comes from providing functionality. To attract contributors, these results suggest that good internal documentation helps. However, to encourage significant additions to functionality, a well-structured design appears to be important. It may be that in choosing a project contributors opt for those that will facilitate their contributions by having good internal documentation enabling them to more easily pinpoint places to make additions and good design features facilitating the addition of functionality instead of modification to improve poor design.

6.6. Limitations

The sample presents possible limitations on the generalizeability of results. By eliminating projects that represented multiple sub-projects, the sample included relatively smaller OSS efforts; large projects such as Linux may follow different patterns. The problem domains selected focused on projects likely to be of most interest to technical audiences; other domains, such as games, could experience different patterns due to the involvement of different kinds of stakeholders. Finally, complexity measures are somewhat language-dependent, and evolutionary paths could be influenced by the choice of programming language.

6.7. Conclusion

The study expanded understanding of software evolution using FDA to uncover three patterns of evolution in size and three different patterns of evolution in structural complexity. A major lesson is that these dimensions of complexity evolve differently. How size evolves in a project is not a good indicator for how structural complexity evolves and vice versa. As expected based on prior work, changes in software size over time were generally positive, but the data yielded a surprising finding showing a significant set of projects for which changes in structural complexity over time were negative, a desirable outcome. In investigating contributing factors and impacts of evolutionary patterns, we found that high initial internal documentation seemed to spur evolution in the projects, and that projects that evolved tended toward a common level of structural complexity. These findings may raise additional questions such as what social factors serve as antecedents or consequences of evolutionary patterns?

7. References

[1] F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, vol. 20, pp. 10-19, 1987.

[2] D. P. Darcy and C. F. Kemerer, "OO Metrics in Practice," IEEE Software, vol. 22, pp. 17-19, 2005.

[3] C. F. Kemerer, "Software Complexity and Software Maintenance: A Survey of Empirical Research," Annals of Software Engineering, vol. 1, pp. 1-22, 1995.

[4] C. K. Prahalad and M. S. Krishnan, "The new meaning of quality in the information age," Harvard Business Review, pp. 109-118, 1999.

[5] D. S. Goldin, "Taming software complexity is critical," in Design News. vol. 56, 2001, p. 172.

[6] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," Management Science, vol. 52, pp. 1015-1031, 2006.

[7] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexty: Implications for software defects," IEEE Transactions on Software Engineering, vol. 29, pp. 297-310, 2003.

[8] E. Barry, C. Kemerer, and S. Slaughter, "How software process automation affects software evolution: a longitudinal empirical analysis," Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, pp. 1-31, 2007.

[9] E. Barry, S. A. Slaughter, and C. F. Kemerer, "An empirical analysis of software evolution profiles and outcomes," in International Conference on Information Systems, Charlotte, North Carolina, USA, 1999, pp. 453-458.

[10] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation," IEEE Transactions on Software Engineering, vol. 32, pp. 365-381, 2006.

[11] W. Scacchi, "Understanding the requirements for developing open source software systems," IEE Proceedings on Software, vol. 149, pp. 24-39, February 2002.

[12] D. P. Darcy, C. F. Kemerer, S. A. Slaughter, and J. E. Tomayko, "The structural complexity of software: An experimental test," IEEE Transactions on Software Engineering, vol. 31, pp. 982-995, 2005.

[13] Y. Wang and J. Shao, "Measurement of the cognitive functional complexity of software," in The Second IEEE International Conference on Cognitive Informatics, 2003, pp. 67-74.

[14] G. Triantafyllos, S. Vassiliadis, and J. G. Delgado-Frias, "Software Metrics and Microcode Development: A Case Study," Journal of Software Maintenance: Reserach and Practice, pp. 199-224, 1996.

[15] B. Kitchenham, L. Pickard, and S. Linkman, "An Evaluation of Some Design Metrics," Software Engineering Journal, 1990.

[16] L. Briand, J. Wust, J. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," Journal of Systems and Software, vol. 51, pp. 245-273, 2000.

[17] N. Gorla and R. Ramakrishnan, "Effect of software structure attributes software development productivity," Journal of Systems and Software, vol. 36, pp. 191-199, 1997.

[18] J. M. Bieman and L. M. Ott, "Measuring Functional Cohesion," IEEE Transactions on Software Engineering, vol. 20, pp. 644-657, 1994.

[19] D. H. Hutchens and V. R. Basili, "System Structure Analysis: Clustering with Data Bindings," IEEE Transactions on Software Engineering, vol. 11, pp. 749-757, 1985.

[20] G. Manduchi and C. Taliercio, "Measuring software evolution at a nuclear fusion experiment site: a test case for the applicability of OO and reuse metrics in software characterization," Information and Software Technology, vol. 44, pp. 593-600, 2002.

[21] A. MacCormack, J. Rusnak, and C. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," Management Science, vol. 52, pp. 1015-1030, 2006.

[22] L. A. Belady and M. M. Lehman, "A model of large program development," IBM Systems Journal, vol. 3, pp. 225-252, 1976.

[23] C. DiBona, S. Ockman, and M. Stone, Open Source: Voices from the Open Source Revolution: O'Reilly and Associates, 1999.

[24] R. Grewal, G. L. Lilien, and G. Mallapragada, "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," Management Science, vol. 52, p. 1043, 2006.

[25] J. O. Ramsey and B. W. Silverman, Applied Functional Data Analysis: Methods and Case Studies. New York: Springer-Verlag, 2006.

[26] K. J. Stewart, D. P. Darcy, and S. L. Daniel, "Opportunities and challenges applying functional data analysis to the study of open source software evolution," Statistical Science, vol. 21, 2006.

[27] W. Jank and G. Shmueli, "Profiling price dynamics in online auctions using curve clustering," Smith School of Business, University of Maryland, College Park 2005.

[28] E. S. Raymond, The Cathedral and the Bazaar: Musing on Linux and Open Source by an Accidental Revolutionary. Sabastopol, CA: O'Reilly, 2001.