Domain Drivers in the Modularization of FLOSS Systems

Andrea Capiluppi
Centre of Research on Open Source Software,
University of Lincoln,
Brayford Campus,
Lincoln,
LN5 7TS,
United Kingdom
acapiluppi@hemswell.lincoln.ac.uk

Abstract. The classification of software systems into types has been achieved in the past by observing both their specifications and behavioral patterns: the SPE classification, for instance, and its further supplements and refinements, has identified the S-type (*i.e.*, fully specified), the P-type (*i.e.*, specified but dependent on the context) and the E-type (*i.e.*, addressing evolving problems) among the software systems.
In order to detect types, and establish similarities, among Free/Libre/Open Source Software (FLOSS) systems, this paper considers three modular characteristics (functions, files and folders) and their evolution: how they are evolving with size, if they are constant across systems, and whether recurring evolutionary patterns are observed. Using these various-grained characteristics, a set of models for the evolution of modularization are extracted from evolving systems, and then used to extract similarities and types from a wide sample of FLOSS projects.
This paper provides three contributions: first, it shows that several models are needed to encompass the variety of modularization patterns; second, it provides three types of models (uni-variate, bi-variate and tri-variate) for the evolution of modularization, with significant goodness-of-fit's. Finally, it shows that two of these patterns alone can interpolate the modular characteristics of the vast majority of a random choice of FLOSS projects.

1 Introduction and Related Work

The classification of software systems into types, if properly conducted, can serve diverse purposes: by classifying and indexing objects or components, for example, developers can ease the search and location of reusable software [8, 21]. As a matter of fact, any reuse effort always involves two major steps to achieve its results: first, it must be able to clearly identify reusable components; second, it must put in place a library of reusable entities for their selection in future projects [15, 16]. As a necessary condition for these two steps, a reusable component should be uniquely classified and described, its functionalities and application domain clearly identified, and its inward and outwards connections described.

Classification of software systems has also been achieved in the past with the purpose of assigning common features of frequently observed patterns to categories, like the SPE program classification [13]. Based on this, a software system can be classified as S-type, when "the specification is the complete, sole and definitive determinant of program properties" [12]; a software system is instead classified as P-type when "the result from the execution of the program is correct in a sense provided by the problem statement" [12]. Given the definition, a P-type system can be considered as S-type as long as its underlying problem was stated completely and precisely. Finally, an E-type system has been described as implementing an application in a "real world domain", and its overall value is only in part dependent on the correctness of the expected outcome, other aspects being the interaction with users and other components. More recently, the same SPE classification was adapted to comprise patterns of behavior and evolution [5].

Finally, from an evolutionary perspective, the classification of software systems has proven useful to identify the presence of patterns of evolution. In recent works, either similarity or volatility, (*i.e.*, variety of behaviors, or absence of similarity) has been used to characterize the evolution of both commercial [1] and FLOSS products [2, 10, 23] and processes [17].

This study builds upon two core concepts of the software engineering knowledge, namely the classification of software systems, as briefly introduced above, and their modularity [19]. The decomposition of software systems into smaller modules, each with large internal cohesion and low coupling with others, is an established framework for software designers and architects. Modules form basic building blocks, and their dimension are typically accomplished avoiding both too large (*i.e.* under-modularization) or too small (*i.e.* over-modularization) components [20].

This paper studies the evolution of three modular characteristics (source functions or methods, source files and source folders) with respect to the size of the system in order both to detect patterns of modular evolution, and to identify clusters or types in the modularization of 26 FLOSS systems evolution. Since types and patterns are here expressed in terms of multivariate models, the paper will initially assess whether a single model could instead be fitted for all the systems. Later, these empirical models will be used to interpolate the modular characteristics of two random samples of FLOSS projects.

This paper is articulated as follows: Section 2 will introduce the definitions and the empirical hypothesis that this study is based upon, since it proposes multiple, multivariate models. Section 3 will test the hypothesis by introducing a cross-sectional analysis: all of the observed systems will be put in the same evolutionary pool, and a single model will be sought. Section 4 will at first discuss the problem of multi-collinearity when dealing with multi-variate models. Then it will investigate and display the models which were extracted from the evolutionary history of the 26 FLOSS systems. Section 5 will use these models to evaluate the modular characteristics of a the second sample of FLOSS systems, and to assess which of these models better formalizes what system. Finally, Section 6 will present the conclusions and illustrate potential future works.

2 Definitions

From the theoretical standpoint, modularity of FLOSS projects has been extensively analyzed [7, 14, 18], and advocated as a necessary condition for appropriately leveraging the distributed approach of FLOSS developers. Less frequently it has been evaluated empirically, and mostly on fine-granular elements: for example, the modularization (in terms of functions) in FLOSS procedural languages has been compared to the modularization (in terms of methods) in FLOSS object-oriented languages [6]; also, the common coupling among modules of the Linux kernel has been extensively analyzed [25].

The terminology and definitions used in this paper are therefore extracted from similar studies in the past FLOSS literature [3, 9, 10, 22], especially those related to entities with different levels of granularity. The empirical hypothesis that this paper is built upon is presented in Section 2.1.

1. Source function: basic unit of source code; this term is used to refer to procedures, subroutines, but also OO-methods.
2. Source file: any file with at least one source function.
3. Source folder: any folder containing at least one source file [4]. The term module is used here to refer to source code functions, files and folders.
4. Size: the length of the whole system, of a folder, a file or a function, which can be evaluated at different levels of granularity, for example: number of folders, of files, lines of code (LOC) and lines of source code (SLOC) which excludes blank lines and embedded comments.
5. Application domain: The application domains of the sample has also been studied. These domains are those used within a well known FLOSS repository (the SourceForge site) to effectively cluster the projects. Table 1 summarizes the domains and the relative keys used throughout the paper.
6. Programming language: this paper will differentiate, for each project, between a procedural (P) and object-oriented (OO) paradigm. This distinction will be made based on a prevalence (*i.e.* more than 80%) of one specific programming language (and paradigm) over others (4th column of Table 2). In cases where multiple programming languages (and paradigms) are present with similar shares, an appropriate notation is used. As an example, the project with Id 5 is composed of C source files (40%) and the rest of the Dylan (also procedural) programming language.

2.1 Working Hypothesis

Previous studies have been conducted to inform about the presence of correlation between

a the size of FLOSS systems and
b time of development (in days, weeks or months)

Table 1. Application domains as used in the SourceForge repository

Application Domain	Key
Communications	A
Database	B
Desktop Environment	C
Education	D
Formats and Protocols	E
Games/Entertainment	F
Internet	G
Multimedia	H
Office/Business	I
Other/Nonlisted Topic	J
Printing	K
Scientific/Engineering	L
Security	M
Software Development	N
System	O
Terminals	P
Text Editor	Q

showing very high goodness-of-fit [2, 10]. It is argued here that these models present severe pitfalls: the set of resulting models not only lacks information on how modularization is achieved, but also establishes a relation between an internal attribute (*i.e.* size, in SLOCs) and an external measurement (*i.e.* time, in days, weeks or months). This paper will explore evolutionary models comprising an internal dependent variable (size), and internal independent variables (number of source folders, files and functions). This will in turn remove any modeling distortions resulting from long periods of inactivity or peaks of activity (as seen in FLOSS systems in proximity of major releases [9]).

The working hypothesis underlying this study states that a single modularization model cannot encompass the variety of FLOSS observed evolutionary patterns. In terms of null hypothesis (H0), the model [m0]

$$size = a * folders + b * files + c * functions + d \text{ [m0]}$$

will produce an adequate goodness-of-fit for all the selected FLOSS projects. The empirical evaluation of this will be achieved analyzing the level of significance of the four parameters (a, b, c and d), *i.e.* evaluating their t-value's and p-value's.

The alternative hypothesis, H1, requires that several models are necessary to fit the modularization patterns of FLOSS projects. As a summary, Table 3 displays the null and the alternative hypotheses, their description, and how they will be tested.

Table 2. Summary of Programming paradigms and Application Domains

Id	Project	Releases	Language	Domain
1	abiword	82	OO/P	Q
2	arla	68	P	G
3	gaim	98	P	A
4	ganymede	42	OO	G
5	gdylan	17	PP	M
6	ghemical	21	OO	L
7	gimp_print	117	OO	K
8	gimp_stable	34	OO	H
9	gimp_dev	96	OO	H
10	gist	19	OO	G
11	grace	36	P	L
12	htdig	17	P	G
13	ksi	14	P	M
14	lcrzo	56	P	G
15	motion	81	P	H
16	mplayer	77	P	H
17	mrtg	77	P	G
18	mutt	91	P	A
19	netwib	35	P	G
20	rrdtool	35	PP	O
21	siagoffice	46	P	I
22	vovida	14	OO/P	A
23	wine_stable	20	P	O
24	wine_unstable	90	P	O
25	xfce	67	P	C
26	xmms	29	P	H

Table 3. Summary of the research hypotheses

Type	**Description**	**Measures**
H0	Single model [m0] for all FLOSS systems	t-value's of a,b,c,d large; p-value's of a,b,c,d ≤ 0.2
H1	Multiple models needed	t-value's of a,b,c,d small; p- value's of a,b,c,d > 0.2

3 Cross-Sectional Analysis

A cross-sectional study design [11] is used in this section for validating and testing the research hypothesis. This type of statistical test is ideal for the proposed hypothesis, since it builds a very basic form of understanding of the data. In this case, it helps in detecting whether a generic, overall model can be established between the dependent variable (size, in LOCs) and the modular characteristics (source folders, files and functions). In a cross-sectional analysis, either the entire population or a subset is selected,

in a single snapshot (*i.e.* no longitudinal analysis is performed): in the case depicted by this paper, the overall population of the 26 FLOSS projects was put in the same statistical pool to detect a unique relationship.

3.1 Design of the Experiment

The purpose of the investigation is to assess the significance of the modularization model [m0]: each of the parameters (a,b,c and d) will be extracted from the data of all the systems, together with its level of confidence, in terms of t-value and p-value. This was repeated several times, in a stratified approach: the steps below summarize the design and implementation of the statistical analysis.

1. At first, the systems in Table 2 were ordered by number of available releases, and a lower limit was set as a minimum to conduct the study: a minimum threshold of 29 releases was selected as the first cross-section (first row of Table 4), as it appeared to be large enough to collect statistical data.
2. All the systems with exactly or more than 29 releases (therefore excluding projects with ID's 5, 6, 10, 12, 13, 22, 23 from Table 2) were listed, and their latest 29 releases, with their data on source folders, files and functions, comprising 551 data points, formed the first population, for which the first multi-variate regression model was calculated. The coefficients of the model, as well as the determination coefficient (R^2) were evaluated (4th, 5th, 6th and 3rd columns of Table 4).
3. As a second step, the number of releases closest and larger than 29 was selected as the next threshold (*i.e.*, 34). As done previously, all the systems with exactly or more than this threshold of releases were considered (hence excluding project with ID 8 from Table 2), forming a pool of 612 data points. As before the coefficients of the multi-variate regression were evaluated, together with the R^2 factor.
4. The same approach was applied, recursively, for the ordered number of releases as cross-sections. A decreasing number of projects participated to the various studies, and different pools of data points (2nd column of Table 4) were considered, as per definition of cross-sectional design.
5. At the end of all the iterations, the mean and the variance of the coefficients were evaluated, and later used to evaluate the t-value and the p-value of each attribute.

3.2 Results of the Cross-Sectional Study

The results of the set of steps as briefly summarized above analysis are displayed in the last four rows of Table 4: the t-value's and p-value's are reported for each of the independent variables (folders, files and functions) and the intercept ("Const"). The only confidence achievable is on the two regressed parameters "Funct" and "Const", while the parameters of "Folder" and "File" have a low t-value and a high p-value.

As per the definitions given in Table 3, it is possible to reject the null hypothesis H0: from the sample of 26 FLOSS projects, it's not possible to extract one single modularization model. The variety of observed behaviors (in terms of modularization) of the selected systems requires a larger set of models: in the next sections, each FLOSS

project will be analyzed to discover one or more patterns of evolution of modularization, and the problem of multi-collinearity will be discussed.

Table 4. Cross-sectional design study – results

Cross-Sections	Data points	R^2	Dir	File	Funct	Const
29	551	0.957	422.08	77.76	32.39	26519.83
34	612	0.957	330.94	70.5	33.48	28873.35
35	595	0.958	627.72	65.36	31.31	25449.24
35	560	0.957	625.78	66.11	31.33	24121.65
36	540	0.958	568.18	60.75	32.28	23600.43
42	588	0.959	477.4	55.43	33.51	21133.54
46	598	0.960	364.32	48.24	35.23	18447.44
56	672	0.965	13.9	32.86	39.18	27244.26
67	737	0.965	-47.38	26.79	40.12	27348.65
68	680	0.988	-1239.44	15.42	50.22	28085.51
77	693	0.986	-930.71	9.45	48.53	24898.57
77	616	0.985	-871.73	9.23	48.01	26336.04
81	567	0.989	-804.14	-4.67	48.89	20642.72
82	492	0.987	-749.78	-5.58	48.16	27414.36
90	450	0.990	107.53	91.9	35.36	18794.66
91	364	0.985	-17.44	24.75	46.41	13399.28
96	288	0.983	32.28	21.87	47.01	7075.48
98	196	0.893	1992.91	-308.17	50.08	30839.27
117	117	0.267	-667.08	-24.08	32.24	56036.34
Mean			12.39	17.57	40.2	25066.35
Variance			765.83	85.06	7.59	9476.44
T-value			0.016	0.207	5.297	2.645
P-value (18 d.f.)			0.987	0.838	0.000	0.017

4 Evolutionary Models

The previous Section 3 showed that a single multi-variate correlation, comprising the characteristics of source folders, files and functions, can not represent, on its own, the modularization patterns of the considered FLOSS projects.

One solution to address this issue would be to refine the model to make it inclusive of all the variations of the ob- served behaviors. Another solution is instead to investigate each project in order to detect the presence of one or more modularization models. This second option is more reasonable, also from previous empirical evidence on FLOSS projects [2, 10, 24], which already shows diverse patterns of evolution.

In the next subsection, the problem of multi-collinearity [11, 24] is investigated in order to detect (if any) the principal modular characteristics of each project's evolution, and to discard the non-relevant ones.

4.1 Addressing Multi-collinearity

In its definition, multi-collinearity is the presence of a significant linear relationship (reflected, for instance, by a large value of R^2) between two or more independent (or explanatory) variables. The presence of multi-collinearity poses serious problems when defining the relevance of a variable into the regressed model: for instance, one could overestimate the weight of a variable even if it was perfectly correlated (*i.e.* superfluous) to another one [24].

In order to refine the single model as expressed above, each system was therefore studied on its own, and the characteristics of source folders, files and functions were taken as independent variables and the size in LOCs as dependent variable. This was repeated in all the releases: finally, the correlation among the independent variables was studied. An acceptable multi-variate regression must have low correlations among the independent variables, and should have high correlation between each independent variable (folder, file, function) and the dependent one (size).

Generating Models from Evolutionary Data For only 4 projects (ganymede, gimp-print, gist, lcrzo) the three modular characteristics are relevant in the evolutionary behavior (*i.e.*, the R^2 between each pair of attributes are all < 0.9), thus generating a $size = f(folders, files, functions)$ pattern. In all the remaining projects, one (or more) multi-collinearity problems were detected, since at least two of the attributes were found to be highly correlated. Therefore, in these systems other patterns were sought: whenever a high correlation was found between two variables, their relevance was questioned compared to the other variables. In cases of evident multicollinearity, one of these variables was dropped.

As an example, the system abiword (Id 1) shows a high correlation between number of folders and number of functions; and between number of functions and number of files: therefore, the variable number of functions was excluded from the model. The model explaining the modularization of the abiword system was chosen to be:

$$size = a * (folders) + b * (files) + d \text{ [m1]}$$

and coded as the $size = f(folders, files)$ pattern in the 5th column of Table 7 (with key m1).

For other projects (gaim, Id 3), no correlation was found to be significant. It was chosen to have a richer set of models in this case: the underlying pattern comprises three models (represented as m3, m4 and m5 in Table 7) which jointly represent the pattern:

$$size = a * (nr\ folders) + d \text{ [m3]}$$
$$size = b * (nr\ files) + d \text{ [m4]}$$
$$size = c * (nr\ functions) + d \text{ [m5]}$$

This complex behavior was coded instead as the

$$size = f(\{folders, files, functions\})$$

pattern in Table 7, meaning that the "size"variable can be explained by any of the "number of files", "number of folders"or "number of functions" alone. We claim that

this pattern achieves the best modularization for FLOSS system: all the dependent variables have the same evolutionary trend, and each can explain alone the growth of size. This behavior is visually depicted (when normalized) in Figure 1 for the Gaim system.

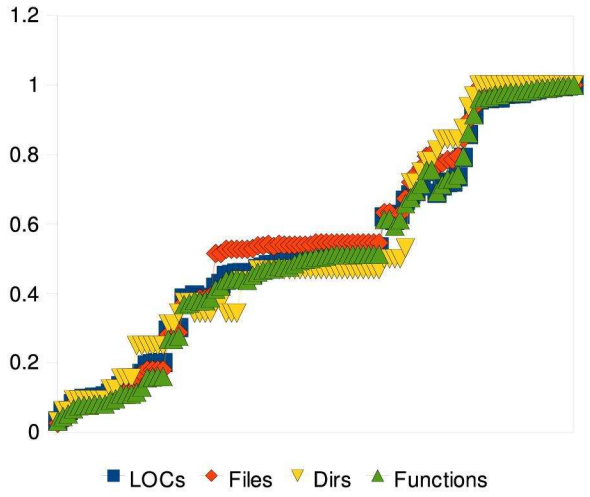

Fig. 1. Evolution trends in the pattern $size = \{folders, files, functions\}$ (Gaim system)

Considering the variations of the 26 patterns, a maximum of 45 different models were found, whose attributes (a, b, c and d) have been evaluated and fully expanded in Table 7. Each of these models was interpolated with evolutionary data in terms of goodness-of-fit. In 6 cases it was found that the corresponding model had a significance lower than 0.9 in terms of R^2: this reduced the overall number of models to 39.

Table 5. Summary – Evolutionary patterns

Pattern	Type of pattern	$R^2 \geq 0.9$
p1	$size = f(\{folder, file, funct\})$	17
p2	$size = f(folder, file)$	12
p3	$size = f(folder, funct)$	7
p4	$size = f(file, funct)$	2
p5	$size = f(folder, file, funct)$	2

In terms of affected patterns, the summary in Table 5 shows the relevance of each pattern as observed in the whole pool, and considering only those cases where the regression fit was larger than 0.9 and 0.7. As observed above, the proposed pattern containing the three explaining variables (p5, in the form of

$$size = f(folders, files, functions)$$

), was empirically observed only in 2 out of 40 statistically relevant patterns (*i.e.*, with $R^2 \leq 0.9$). Most of the other patterns represent an interaction of at most two variables (21 out of 39) or a single variable (17 cases out of 39). Also, the union of the patterns p1 and p2 alone is responsible for some 3/4 of the extracted models.

5 Testing the Models

The relations obtained in the previous Section 4 formed a pool of models, each characterizing the evolution of the size (in LOCs) as a function of the modular characteristics (source folders, files and functions) of a specific project. The goodness of fit of most of these models was demonstrated to be statistically significant via the indicator R^2.

5.1 Design of the Experiment

In this subsection, the models defined above (from m0 to m40) are used to test whether a generic FLOSS project shows a modularization which can be interpolated by any of the models above.

In order to do so, a random sample of 50 FLOSS projects was extracted from both the Debian and the SourceForge repositories, resulting in 100 projects. Debian contains a popular FLOSS forge which is the basis of the successful Debian distribution; SourceForge hosts more than 200,000 FLOSS projects and is recognized as the most common FLOSS portal. The two samples were extracted from similar-sized pools, *i.e.* the "stable" subset of hosted projects, as per the development status that each FLOSS project can select as its own. For each project in the samples, the number of folders, files and functions were evaluated together with the value of size in SLOCs.

Table 8 shows the summaries of the attributes of these samples: information on the "programming languages" and "application domains" was also collected: depending on the programming language, OO projects was selectively interpolated with OO models, and similarly for procedural projects. For some of the projects, information on source functions could not retrieved, since their languages (like Python, PHP or the such) were not supported by the available tools. These projects were discarded from the model testing.

The modular characteristics of each project were used within a model obtaining an estimated size as the dependent variable: this value was later compared with the real value of size (in SLOCs). The error made by the estimation was evaluated as follows

$$error = abs\left[\frac{Size_{re} - Size_{est}}{Size_{re}}\right]$$

where $Size_{re}$ represents the real size value, and $Size_{est}$ the estimate given by the model. The p25 (the probability that the estimate diverges for less than 25% from the real value) is recorded in Table 6. The summary differentiates among the application domains of Table 1 and the patterns (p1 to p5) of Table 5: within the projects with application domain "A" (Communication), the first row summarizes that:

- 4 models from Table 7 and pattern p1, *i.e.* f(folder, file, function), estimate the achieved size of a subset of projects (sharing the same programming paradigm of these models – OO vs P) with an error $< 25\%$;
- only one model with the pattern p2 (f(folder, file)) can properly estimate the size in SLOCs of one project with a similar error;
- 2 models with pattern p3 estimate the SLOCs of the two random samples with a comparable error.

Table 6. Predictability of patters at $p < 0.25$

Pattern	p1	p2	p3	p4	p5
A	4	1	2		
B		2	1		
C	5	1	1		
D					
E			1		
F	4	3	2	1	1
G	11	3	2	1	2
H	10	5	3	1	1
I	9	5	2	1	1
J					2
K	14	4	2	1	2
L	16	2	2	1	
M		2			
N	1	1	1		
O	3	1	1	1	
Totals	**77**	**30**	**20**	**7**	**9**
%	**53.85**	**20.98**	**13.99**	**4.90**	**6.29**

5.2 Results of the Experiment

The following insights can be drawn from this summary table:

1. As visible in the last two rows, two patterns perform better than others in terms of error made in the estimation of size (in SLOCs). Alone, the patterns p1 and p2 cover 34 of the successful estimates: this means that the models based on one attribute alone (*i.e.*, number of folders, or files, or functions) explain the modularization patterns of the majority of the projects in these samples. The size in SLOCs is therefore predictable using just one attribute, and the ratio SLOCs/attribute represents a constant.
2. A subset of application domains (B, D, E, J, M, N) is more difficult to estimate than others. This has two explanations: first, apart from the domain "M", the evolutionary sample of Table 2 does not contain projects belonging to this subset.

Hence, specific models for these domains were not produced. Second, the two random samples of FLOSS projects also contain few projects (6 overall) belonging to these domains, hence making it difficult to draw conclusions on them.

3. A subset of application domains (F, G, H, I, J, L) is instead attracting several estimates from diverse patterns (although the patterns p1 and p2 still prevail). Apart from the "G" (Internet) domain, they all represent front-end user applications (as opposed to back-end system administrators): this result is therefore stating that the projects in these domains have diverse modularization types, ranging from highly modular (as mirrored by the p1 pattern) to the least modular, when the three modular characteristics (folder, file and function) are needed to estimate the size in SLOCs (as mirrored by the pattern p5).

6 Conclusions

This paper used publicly available FLOSS data and shared metrics in order to provide a mechanism to classify the evolution of software systems. In past literature, the shapes of evolutionary curves have been qualitatively observed, or univariate models (size-time) have been used to draw similarities among systems. In this study, the relationship between size (as dependent variable) and the modularization characteristics of systems (number of source folders, files and functions – as independent variables) was used to first extract models of evolution, select patterns out of these and then fit these modularization models on a random sample of FLOSS projects.

The study was preceded by a research hypothesis: a unique modularization model can not capture the variety of observed behaviors in FLOSS systems, but a set of modularization models is needed. The model (m0) used as a benchmark was a multi-variate linear correlation, in the form $size = a*folders + b*files + c*function + d$. This hypothesis was tested through a cross-sectional analysis, using the whole set of gathered FLOSS data (summing up to more than 1,300 data points). It was found that the single model could not be considered statistically accurate for each and all the considered systems.

As a result of the hypothesis testing, the presence of a whole set of modularization models was investigated. For every FLOSS project in the pool, the benchmark model m0 was analyzed, and the multi-collinearity issue was discussed: for some of the projects, in fact, the multivariate model revealed the presence of collinearity among some of the independent variables. In those cases, a simpler pattern was tested, either with only two independent variables, or just with one. In some of the projects, the multi-collinearity of the variables pointed to a complex pattern, where more than one patterns were used to describe the evolution of modularization. As a result of this step, it was observed a dominance of univariate and bi-variate patterns over the benchmark model, which could be observed as statistically relevant (in terms of R^2) only in 2 out of 39 models.

In order to test these models, and their accuracy in predicting the modularization of FLOSS systems, a random sample of FLOSS projects was extracted from the Debian distribution and the SourceForge portal, and their modular characteristics recorded.

The models were used to interpolate the variables, and to predict the size, of the FLOSS projects: models with a specific programming paradigm (OO or procedural) were used to interpolate the FLOSS projects using the same paradigm. This prediction was then compared to the actual size in SLOCs of the project, with an error of 25%.

The first finding of this analysis showed that two patterns stand out in terms of prediction power: p1 and p2 could cover up to 3/4 of the successful predictions. The second finding pointed at the uneven distribution of domains in a random sample, showing that specialized topics (Databases, Education) are also more difficult to model. The third finding showed that high-end applications suffer from a high variety of modularization patterns, ranging from very modular models (where each attribute can be considered as constantly growing with the size) to uneven growth of each attribute, resulting in a model where each attribute is needed to interpolate the achieved size.

References

1. Barry, E.J., Kemerer, C.F., Slaughter, S.A.: On the uniformity of software evolution patterns. In: ICSE '03: Proceedings of the 25th International Conference on Software Engineering, pp. 106–113. IEEE Computer Society, Washington, DC, USA (2003)
2. Capiluppi, A.: Models for the evolution of os projects. In: ICSM '03: Proceedings of the International Conference on Software Maintenance, p. 65. IEEE Computer Society, Washington, DC, USA (2003)
3. Capiluppi, A., Boldyreff, C.: Identifying and improving reusability based on coupling patterns. In: ICSR '08: Proceedings of the 10th international conference on Software Reuse, pp. 282–293. Springer-Verlag, Berlin, Heidelberg (2008). DOI 10.1007/978-3-540-68073-4_31
4. Capiluppi, A., Morisio, M., Ramil, J.F.: The evolution of source folder structure in actively evolved open source systems. In: METRICS '04: Proceedings of the Software Metrics, 10th International Symposium, pp. 2–13. IEEE Computer Society, Washington, DC, USA (2004). DOI 10.1109/METRICS.2004.40
5. Cook, S., Harrison, R., Lehman, M.M., Wernick, P.: Evolution in software systems: foundations of the spe classification scheme: Research articles. J. Softw. Maint. Evol. **18**(1), 1–35 (2006). DOI 10.1002/smr.v18:1
6. Ferrett, L.K., Offutt, J.: An empirical comparison of modularity of procedural and object-oriented software. In: ICECCS '02: Proceedings of the Eighth International Conference on Engineering of Complex Computer Systems, p. 173. IEEE Computer Society, Washington, DC, USA (2002)
7. Fitzgerald, B.: A critical look at open source. Computer **37**(7), 92–94 (2004). DOI 10.1109/MC.2004.38
8. Frakes, W.B., Pole, T.P.: An empirical study of representation methods for reusable software components. IEEE Trans. Softw. Eng. **20**(8), 617–630 (1994). DOI 10.1109/32.310671
9. German, D.M.: Using software trails to reconstruct the evolution of software: Research articles. J. Softw. Maint. Evol. **16**(6), 367–384 (2004). DOI 10.1002/smr.v16:6
10. Herraiz, I., Gonzalez-Barahona, J.M., Robles, G.: Towards a theoretical model for software growth. In: MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories, p. 21. IEEE Computer Society, Washington, DC, USA (2007). DOI 10.1109/MSR.2007.31

11. Lauridsen, J., Mur, J.: Multicollinearity in cross-sectional regressions. Journal of Geographical Systems **8**(4), 317–333 (2006). URL http://ideas.repec.org/a/kap/jgeosy/v8y2006i4p317-333.html
12. Lehman, M.M.: Uncertainty in computer application and its control through the engineering of software. Journal of Software Maintenance **1**(1), 3–27 (1989). DOI 10.1002/smr.4360010103
13. Lehman, M.M., Belady, L.A. (eds.): Program evolution: processes of software change. Academic Press Professional, Inc., San Diego, CA, USA (1985)
14. Lerner, J., Tirole, J.: Some simple economics of open source. The Journal of Industrial Economics **L**(2), 197–232 (2002). URL http://www3.interscience.wiley.com/cgi-bin/fulltext/118942767/PDFSTART
15. McClure, C.: Software reuse techniques: adding reuse to the system development process. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1997)
16. Morisio, M., Ezran, M., Tully, C.: Success and failure factors in software reuse. IEEE Trans. Softw. Eng. **28**(4), 340–357 (2002). DOI 10.1109/TSE.2002.995420
17. Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y.: Evolution patterns of open-source software systems and communities. In: IWPSE '02: Proceedings of the International Workshop on Principles of Software Evolution, pp. 76–85. ACM, New York, NY, USA (2002). DOI http://doi.acm.org/10.1145/512035.512055
18. Narduzzo, A., Rossi, A.: The role of modularity in free/open source software development. In: S. Koch (ed.) Free/Open Source Software Development, pp. 84–102. Idea Group Publishing, Hershey, PA (2004)
19. Parnas, D.L.: On the criteria to be used in decomposing systems into modules pp. 139–150 (1979)
20. Pressman, R.S.: Software Engineering: A Practitioner's Approach. McGraw-Hill Higher Education (2001)
21. Prieto-Díaz, R.: Implementing faceted classification for software reuse. Commun. ACM **34**(5), 88–97 (1991). DOI http://doi.acm.org/10.1145/103167.103176
22. Robles, G., Amor, J.J., Gonzalez-Barahona, J.M., Herraiz, I.: Evolution and growth in large libre software projects. In: IWPSE '05: Proceedings of the Eighth International Workshop on Principles of Software Evolution, pp. 165–174. IEEE Computer Society, Washington, DC, USA (2005). DOI 10.1109/IWPSE.2005.17
23. Smith, N., Capiluppi, A., Ramil, J.F.: A study of open source software evolution data using qualitative simulation. Software Process: Improvement and Practice **10**(3), 287–300 (2005). DOI 10.1002/spip.230
24. Trochim, W.: The Research Methods Knowledge Base, Second Edition, 2nd edn. Atomic Dog Publishing, Cincinnati, OH. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/1931442487
25. Yu, L., Chen, K.: Categorization of common coupling and its application to the maintainability of the linux kernel. IEEE Trans. Softw. Eng. **30**(10), 694–706 (2004). DOI 10.1109/TSE.2004.58. Member-Schach,, Stephen R. and Member-Offutt,, Jeff

Appendix

Table 7. Summary – Evolutionary Models

Id	Project	R^2	Model
m1	abiword	0.97	y = 1009.771*folder + 325.588*file - 210142.563
m2	arla	0.99	y = 538.466*folder + 216.124*file - 15779.445
m3	gaim	0.97	y = 6401.148*folder + 8056.888
m4	gaim	0.99	y = 423.250*file + 664.207
m5	gaim	1	y = 38.941*funct + 6219.853
m6	ganymede	0.99	y = 6013.705*folder - 16.579*file + 18.290*funct - 18579.671
m7	gdylan	0.99	y = 262.675*file + 12.387*funct + 26653.283
m8	ghemical	0.99	y = -6665.506*folder + 595.308*file + 2807.564
m9	gimp_stable	0.99	y = -2249.964*folder + 156.274*file + 534680.665
m10	gimp_stable	1	y = -380.309*folder + 48.323*funct + 66610.831
m11	gimp_unstable	1	y = 59.860*folder + 53.262*funct - 28040.954
m12	grace	0.97	y = -5093.335*folder + 622.458*file + 40515.292
m13	htdig	0.99	y = 136.588*folder + 54.784*funct + -14221.918
m14	ksi	0.93	y = 346.067*file - 980.292*folder + 36294.173
m15	htdig	0.99	y = 50.658*file + 48.669*funct - 14224.314
m16	lcrzo	0.97	y = 5173.712*folder + 161.054*file + 19.255*funct - 12490.114
m17	motion	0.95	y = 0.000*folder + 77.202*funct -1174.557
m18	mplayer	0.99	y = -5057.409*folder + 678.961*file - 3626.076
m19	mrtg	1	y = 81.736*folder + 25.549*funct + 2218.992
m20	mutt	0.91	y = -1478.851*folder + 488.015*file + 6342.068
m21	mutt	0.97	y = 559.707*folder + 71.766*funct + -602.318
m22	netwib	0.95	y = 3426.807*folder + 76321.231
m23	netwib	0.99	y = 199.451*file + 71239.165
m24	netwib	1	y = 61.597*funct + 16614.239
m25	rrdtool	0.95	y = 8463.757*folder + 3154.039
m26	rrdtool	1	y = 1173.715*file + -39939.284
m27	rrdtool	0.99	y = 230.029*funct - 47495.961
m28	siagoffice	0.99	y = 1241.303*folder + 311.820*file - 15036.930
m29	vovida	0.98	y = -1244.999*folder + 344.516*file - 13420.463
m30	wine_stable	0.95	y = 4470.605*folder + 384237.197
m31	wine_stable	0.99	y = 777.483*file - 254581.635
m32	wine_stable	0.99	y = 36.475*funct + 189872.728
m33	wine_unstable	0.99	y = 5262.688*folder + 22096.140
m34	wine_unstable	0.99	y = 721.482*file - 157662.177
m35	wine_unstable	1	y = 44.415*funct - 39778.746
m36	xfce	0.98	y = 4667.436*folder + 8039.892
m37	xfce	0.95	y = 131.795*funct - 73091.477
m38	xmms	0.94	y = -1525.592*folder + 665.040*file - 39494.842
m39	xmms	0.97	y = -560.728*folder + 83.277*funct - 27330.425

Table 8. SourceForge sample – modular characteristics (part 1)

	SLOCs	Fold	File	Funct	Domain	Lang
perpojo	1,677	10	31	117	A	OO
moses	105,955	0	1,042	4,053	A	OO
fn-javabot	10,142	35	211	279	A	OO
ozone	63,790	141	1,018	3,920	B	OO
xqilla	107,320	58	824	2,534	B	OO
fsdb	241,218	362	1,715	8,506	B	OO
galeon	93,374	11	412	3,525	C	P
whiteboard	4,910	2	13	202	D	php
fourever	15,163	28	207	593	E	OO
hge	45,654	19	110	800	F	P
zmpp	15,502	24	184	1,063	F	OO
sudapix	234	8	111	15,747	F	P
symbolica	2,623	5	32	67	F	OO
icsDrone	1,411	1	14	33	F	P
kpictorial	21	3	27	18,214	F	sh
critical_care	38,994	18	185	1,051	F	OO
ogce	350,490	1,385	3,222	13,960	G	OO
cpia	22,954	6	25	109	G	P
mod_aspdotnet	2,445	4	13	45	G	OO
xmlnuke	57,944	33	395	1,623	G	php
wxactivex	3,264	1	11	37	G	OO
tab-2	19,067	63	334	597	G	php
source	8,786	128	109	162	G	OO
oliver	1,429	2	21	9	G	php
formproc	3,514	11	70	134	G	OO
freemind	28,519	30	241	1,579	H	OO
cdlite	1,116	1	6	29	H	OO
audiobookcutter	4,229	8	37	34	H	OO
edict	2,556	1	2	0	J	perl
qlc	26,452	10	203	890	J	OO
swtjasperviewer	3,214	4	43	129	K	OO
QPolymer	86,971	7	199	652	L	OO
expreval	3,588	2	18	66	L	OO
eas3pkg	43,724	5	101	69	L	f90
neocrypt	2,135	3	27	21	M	OO

Table 9. SourceForge sample – modular characteristics (part 2)

	SLOCs	Fold	File	Funct	Domain	Lang
juel	7,284	15	110	404	N	OO
csUnit	16,241	41	234	96	N	cs
j_trac	519	34	157	12,771	N	OO
fitnesse	39,503	37	631	2,321	N	OO
ustl	11,416	2	94	684	N	OO
txt2xml	1,345	9	25	61	N	OO
gvision	101,123	9	236	0	N	pascal
seagull	54,155	102	362	878	N	OO
clinkc	25,846	140	432	919	N	P
simplexml	1,691	3	4	65	N	P
pf	213	33	166	84,489	O	perl
Beobachter	2,715	14	49	94	O	OO
blob	22,056	15	276	496	O	P
intermezzo	34,792	15	167	522	O	P
cotvnc	37,455	2	225	789	O	P

Table 10. Debian sample – modular characteristics (part 1)

	SLOCs	Fold	File	Funct	Domain	Lang
kphoneSI	41,829	10	263	735	A	OO
sylpheed	106,087	6	249	2,859	A	P
enigmail	10,790	13	53	86	A	OO
synce-kde	21,684	6	95	141	C	sh:
txt2html	3,623	2	3	0	E	perl:
scid	89,402	6	151	1,179	F	tcl:
netpanzer	74,368	42	598	2,935	F	OO/P
boson	224,567	78	1,272	9,246	F	OO
gosa	107,798	101	466	2,404	G	php:
lirc	44,753	26	148	785	G	P
openh323	234,285	30	451	6,392	G	OO
openafs	618,553	195	2,452	10,807	G	P
peercast	22,543	8	95	818	G	OO
slrn	42,993	5	91	1,189	G	P
cherokee	54,229	17	432	1,221	G	P
vlc	401,256	129	1,378	6,250	H	P
cdparanoia	9,182	3	37	211	H	P/P
kmouth	5,240	3	41	99	H	OO
rlplot	69,493	1	27	1,449	H	OO
flac	56,293	42	206	1,380	H	P

Table 11. Debian sample – modular characteristics (part 2)

	SLOCs	Fold	File	Funct	Domain	Lang
gwenview	4,580	4	62	128	H	OO
prcs1	37,360	8	130	663	H	OO/P
yaml4r	10,728	8	31	0	I	ruby:
xmakemol	18,724	1	39	315	I	P
octave_forge	78,150	129	409	0	I	OO/P
myphpmoney	19,434	11	64	153	I	php:
dia	146,550	43	561	4,151	I	P
grass6	107,648	115	558	1,650	I	P
geomview	101,844	86	771	2,748	I	P
ProofGeneral	48,692	22	134	0	I	lisp:
fte	51,498	2	186	1,182	K	OO
ruby	419,942	260	2,076	5,086	K	ruby:
EtoileWildMenus	1,711	1	21	2	K	OO
tcl	165,306	23	378	2,205	K	P
wxWidgets	2,142,713	372	4,325	0	K	OO
libax25	11,721	1	30	80	K	sh:
liboil	52,996	39	304	730	K	P
libsoup	15,012	3	86	494	K	P
Pike	173,196	62	408	2,302	K	P
shorewall	25,159	6	74	0	L	sh:
acpidump	2,349	1	16	53	L	P
tiobench	1,689	1	8	41	L	P
radiusd	95,967	101	397	1,330	L	P
preludemanager	10,854	15	70	304	L	P
apmud	2,502	1	14	45	L	P
clamav	116,731	24	339	1,056	L	P
tdb	3,942	3	19	133	L	P
grub	3,536	1	7	0	L	sh:
noteedit	63,456	3	139	611	M	OO
jToolkit	4,156	5	32	0	M	python: