A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects

Paulo Meirelles, Carlos Santos Jr., João Miranda, Fabio Kon
FLOSS Competence Center
Institute of Mathematics and Statistics
University of São Paulo, Brazil
(CCSL-IME/USP)
{paulormm,denner,joaomm,fabio.kon}@ime.usp.br

Antonio Terceiro, Christina Chavez
Department of Computer Science
Federal University of Bahia, Brazil
(DCC-UFBA)
{terceiro,flach}@dcc.ufba.br

***Abstract*—A significant number of Free Software projects has been widely used and considered successful. However, there is an even larger number of them that cannot overcome the initial steps towards building an active community of users and developers. In this study, we investigated whether there are relationships between source code metrics and attractiveness, i.e., the ability of a project to attract users and developers. To verify these relationships, we analyzed 6,773 Free Software projects from the SourceForge.net repository. The results indicated that attractiveness is indeed correlated to some source code metrics. This suggests that measurable attributes of the project source code somehow affect the decision to contribute to and adopt a Free Software. The findings described in this paper show that it is relevant for project leaders to monitor source code quality, particularly a few objective metrics, since these can have a positive influence in projects chances of forming a community of contributors and users around their software, enabling further enhancement in quality.**

I. Introduction

The adoption of Free and Open Source Software[1] has significantly increased in the last decades to the point of becoming influential to the global economy [1]. Although Free Software has emerged as a movement supported by volunteer developers, many large companies are now involved in it [2], [3]. According to a Forrester Consulting survey, which compared large companies in Europe and North America [4], the usage of Free Software is currently widespread in the back-end, middleware, office productivity tools, and business applications software categories. Moreover, this survey states that 92% of the senior business and IT executives say that Free Software products have met and, in some cases, exceeded their quality expectations.

This satisfaction and quality is usually achieved thanks to the collaboration of a large user and developer community who reports failures, fixes bugs and adds features. In fact, the Free Software development model is said to offer two main advantages: the potential for peer-review and the possibility of attracting developers from different parts of the world [5]. Hence, an important issue for Free Software projects is to attract volunteers [6].

[1]In this study, we consider the terms Free Software and Open Source Software (OSS) equivalent.

However, not all Free Software projects reach success and high quality [5]. The amount of inactive projects is undoubtedly higher compared to the number of active projects. To illustrate this scenario, consider the data extracted in November, 2009, from Sourceforge.net, one of the most popular Free Software repositories. Out of its 201,494 projects, only 60,642 had more than one release, 40,228 had been downloaded more than once, and 23,754 had more than one member. Finally, only 12,141 projects matched all these criteria simultaneously. This may indicate that no more than 6% of the projects on SourceForge.net are able to have a healthy community of users and developers benefiting from a Bazaar style of development [7].

Santos Jr. *et al.* [8] defined a theoretical model for attractiveness as a crucial construct for Free Software projects, proposing their (i) typical origins (e.g., license type and intended audience); (ii) indicators (e.g., number of members and downloads); (iii) consequences (e.g, levels of activity and time for task completion). They suggested that the success of any project depends on its level of attractiveness to potential contributors and users. Based on this model, our study explored some of the factors that may enable projects to build a community by attracting users and developers. Specifically, our focus rests on objective factors: we investigated whether attractiveness can also be influenced by measurable source code attributes – structural complexity and size.

Although source code metrics have been proposed since the 1970s [9], their potential use as guidelines for software development has not been fully explored yet [10]. In particular, we have observed that many Free Software projects do not practice source code quality evaluation and have no tools available to do so. This lack of systematic code evaluation leaves a lot of room for improvement in Free Software projects' processes and practices [5].

In general, despite the high importance of source code in the Free Software community, called the "show me the code" culture, source code metrics are often not perceived as an indicator of quality. To address this apparent contradiction, we argue, theoretically, that source code metrics are related to project attractiveness and, thus, influence its success. To verify these ideas empirically, we analyzed 6,773 projects written in the C language from SourceForge.net. We show that, considering such a sample, one structural complexity metric and two size metrics play an important role in explaining attractiveness, here represented by the number of downloads and the number of project members.

The remainder of this paper is organized as follows: Section II presents the theoretical foundations of source code metrics and attractiveness. Section III presents our hypotheses and shows the selection criteria and definition for the variables used in our study. Section IV evaluates the hypotheses and discusses their results. Section V reviews related work. Conclusions and future research directions are discussed in Section VI.

II. Theoretical Background

In this section we present the definitions of the selected source code metrics (Section II-A) and the attractiveness concept, including its proxies (Section II-B). These were chosen to build a statistical model that represents the relationships proposed.

A. Source code metrics

For our subset of source code metrics, we used the "module" concept as a general term for the different types of structures used in software development. Therefore, it stands for classes, abstract data types and source files. Specifically in this study, we consider a C source file as a "module".

Similarly, we generalized the concept of "method" to "function", denoting a portion of source code that performs a specific task.

The most commonly used metric to measure software size is *Lines of Code* (LOC), which indicates the number of non-blank and non-comment source code lines. Using the LOC metric as a basis for comparison between projects requires the projects to be written in the same programming language [11].

On the other hand, *Number of Modules* is another useful size indicator, which is somewhat less influenced by programming languages and line-level coding styles. With that being said, it can be used to compare projects written in different languages [10].

When considering characteristics such as maintainability, flexibility, comprehension effort, and source code quality in general, one has to take into account not only the size metrics described above but also structural metrics, such as the ones that follow.

Number of Functions (NOF) is used to measure module size in terms of the operations it implements. This metric is used to help identify the reuse potential of a module. In general, modules with a large number of functions are more difficult to reuse because they tend to be less cohesive [12]. Hence, a module should not have an excessive number of functions [13].

Number of Public Variables (NPV) and *Number of Public Functions* (NPF) are metrics related to module encapsulation. They measure the potential communication among modules [14]. Once good programming practices recommend that module variables should be only manipulated via accessor functions [13], module variables should be private, indicating that the optimal number for this metric is zero. The number of public functions in a module represents the size of the "module interface". Functions are directly related to the operations provided by their module. High values for this metric indicate that a module has a lot of functions and, probably, many responsabilities, conflicting with good programming practices [13].

Cohesion is a measure of the diversity of "topics" that a module implements. High cohesion values indicate whether a module focus on a single aspect of the system, while low indicates it deals with several different aspects. In terms of undestanding, modification and maintainability, highly cohesive modules should be seeked. A metric commonly used for cohesion is *Lack of Cohesion of Methods* (LCOM), originally proposed by Chidamber and Kemerer [15]. High LCOM values indicate low cohesion, while low LCOM values indicate high cohesion.

The first LCOM definition, called LCOM1, corresponds to the number of function pairs of a module which manipulate the same module variables. Once it has received several criticism and revision proposals, through this study we used the revised definition by Hitz and Montazeri, known as LCOM4 [16]. In order to calculate LCOM4 of a module, it is necessary to build an undirect graph in which the nodes are its functions and variables. For every function, there should be an edge between it and another function or variable it uses. The LCOM4 value is the number of weakly connected components of this graph.

Coupling is a measure of how one module is connected to other modules in the project. High coupling indicates a greater difficulty to change the modules of the system, since a change in one module may have an impact in all other modules that are coupled to it. In other words, if coupling is high, the software tends to be less flexible, more difficult to adapt, and more difficult to understand. A straightforward method to measure coupling is *Coupling Between Objects* (CBO), once again proposed by Chidamber and Kemerer. CBO measures how many modules are used by the one being analyzed [15].

The more complex a piece of software, the more challenging it is to change and evolve it. Coupling and cohesion have been described and discussed in many works as essential indicators of structural complexity [17]. Moreover, it is widely known that to build high-quality and flexible software, it is advisable to seek low coupling and high cohesion [18].

In fact, Darcy *et al.* [17] showed that, individually, neither coupling nor cohesion are related to a software maintenance effort. Both must be considered together. When combined, the product of coupling and cohesion as a metric is positively correlated to the maintenance effort.

In conclusion, seven source code metrics, selected according to criteria shown in Section III, were used on the statistical model for our study. In particular, we use the product of coupling (CBO) and cohesion (LCOM4) as our metric of structural complexity (SC) [17].

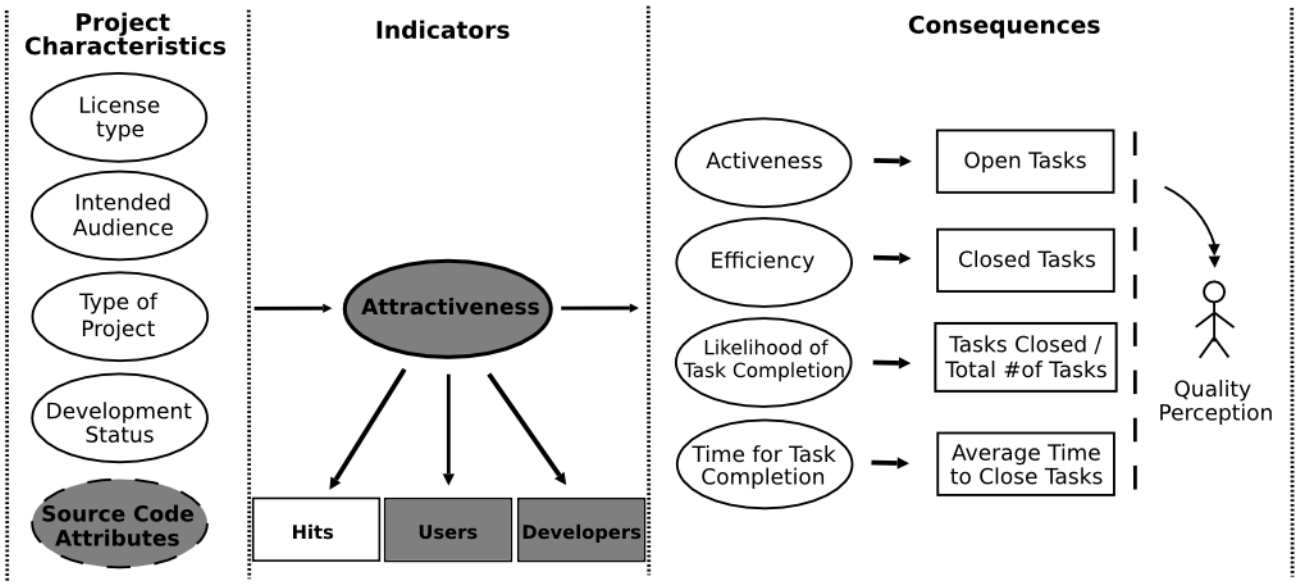

Fig. 1. Attractiveness research model – adapted from Santos Jr. et al [8].

B. Attractiveness

Attractiveness is the capacity of bringing users and developers to a project. A Free Software project is as attractive as it has the ability to be appealing to potential users and developers. They will later use the software and, ultimately, participate on tasks to improve the project [8].

In our study, the concept of attractiveness and its proxies are based on Santos Jr.'s attractiveness model [8], one of our previous works. In that work, we presented a research model for attractiveness, shown in Figure 1. We now intend to expand and adapt it to our hypotheses about source code attributes and attractiveness, as emphasized also in Figure 1. This model specifies project characteristics that influence its attractiveness, and the consequences of attractiveness (e.g., levels of activity, efficiency, likelihood of task completion, time for task completion, and quality perception) [8].

Originally, the project characteristics proposed were:

- *License Type* under which a program is available, such as GPL, BSD, and Mozilla Public. The license influences the use and distribution of a project, and defines the rules for creating derivative works, regulating what can and cannot be done with the source code [8]. These restrictions impact people's motivations to use and develop Free Software.
- *Intended audience* is the type of users (e.g., beginner, advanced) and members (e.g., system administrator, Java programmers) a project aims at. Audience can influence the number of potential developers for it expands or shrinks the target population size [19]. Moreover, specific types of users attract specific members, which define their expertise and likelihood to contribute [8].
- *Type of project* refers to the specific area to which a project is related, such as genealogy, payroll, browsing, games, scientific, etc [20]. This represents the application domain and influences attractiveness for marketing reasons normally known as niches. Some niches have more volunteer labor (and user base) available than others. Moreover, some application domains have more competing (similar) projects than others, making it harder for a project to stand out as a viable option for use [19].
- *Development status* – life-cycle status – refers to the current available versions of the software. This could be, for example, testing, beta, stable, production, and mature. This status can influence a developer's decision to join and contribute to a project, as well as a user's decision to adopt its software [8]. It also can affect members' motivations to work in order to release a new version, affecting productivity rates [21], [6].

We propose to insert source code attributes as a project characteristic in this theoretical cause and effect model of Free Software attractiveness, shown in Figure 1. To achieve this, we defined a simple (intermediary) attractiveness model to observe the individual influence of source code attributes on Free Software attractiveness – a subset of variables highlighted in Figure 1. In short, we propose a new element that can explain attractiveness partially. We did not deal with the consequences to attractiveness when we added source code attributes as an influence. However, we expect that they would work in the same causal chain manner as shown in Figure 1.

Before a Free Software project can receive failure reports, bug fixes, and new features, it should be attractive to volunteers, who normally first use and join the project, providing contributions later. Over time, these contributions affect the number of downloads and bring more members, creating a positive feedback loop. Thus, we estimated attractiveness based on two of its empirical indicators:

1) *number of downloads* as registered for SourceForge.net

projects, which represents the number of people interested in using the software.

2) *number of members* as registered for SourceForge.net projects, which represents the number of contributors to the project.

One should note that the numbers of downloads and members at SourceForge.net are proxies to the actual numbers of users and developers of the project, respectively. This study explored a large number of projects and applied the same criteria uniformly to all of them. Although we recognize that using these proxies is a limitation, we did not find in previous works better proxies that could represent more faithfully the number of users and number of developers in a large sample of projects.

The meaning of "success" to Free Software projects was discussed from different perspectives in previous research: (i) source code modularity [22]; (ii) number of lines of code generated [23]; (iii) velocity of closing bugs [6]; (iv) ability of a project to advance through development phases (e.g., from alpha to beta to stable) [21], [20]; (v) number of downloads [24]; (vi) number of members [25].

In our understanding, these measures, when individually used, do not indicate fully a successful Free Software project; but, when analyzed together, they offer the means to help achieve success, or keep it [8]. Additionally, the vast majority of collaboration for a Free Software project lies on its source code, which is the most important "artifact" generated and managed by and for its community. Therefore, we propose to insert in the attractiveness model the source code attributes – obtained for source code metrics – as one of the attractiveness origins. In summary, some source code characteristics can lead to more contributions for a project, which may attract more users and developers – our hypothesis are based on this idea.

III. Research Design

Initially, this Section presents a source code analysis tool under development by our group called `Analizo` (Section III-A). It was used to calculate source code metric of 6,773 Free Software projects from SourceForge.net. Later, Section III-B presents the criteria used for sample and data collection. Finally, Section III-C shows the multiple regression model defined to test our hypothesis, which are discussed in Section III-D.

A. The Analizo Tool

`Analizo`[2] is a multi-language source code analysis tool. Its architecture was designed to support source code parsing in different languages and to report useful information about it.

A basic requirement of our source code analysis tool was the ability to analyze source code written in multiple languages. Most existing tools use object code to extract data, making it impossible to process projects that do not compile due to failures in either the source code or in its dependencies [26]. In addition, tools based on object code are not capable of analyzing features present only in the source, such as comments. To avoid these problems, `Analizo` is designed to extract the information directly from source code files.

`Analizo` uses `Doxyparse`[3], a multi-language source code parser based on Doxygen's internals, to parse the source code. This feature provides `Analizo` with the potential ability to parse all the languages supported by `Doxygen`[4]. So, up to the moment, it has been tested only with C, C++, and Java source code, supporting the computation of fourteen metrics:

- *Afferent Connections per Class* (ACC),
- *Coupling between Objects* (CBO),
- *Coupling Factor* (COF),
- *Depth of Inheritance Tree* (DIT),
- *Lack of Cohesion on Methods/Functions* (LCOM4),
- *Lines of Code* (LOC),
- *Lines per Method/Function* (AMZ_Size),
- *Number of Attributes/Variables* (NOV),
- *Number of Children per Class* (NOC),
- *Number of Methods/Functions* (NOF),
- *Number of Classes/Module* (NM),
- *Number of Public Attributes/Variables* (NPV),
- *Number of Public Methods/Functions* (NPF),
- *Response for Class* (RFC).

The correctness of the metrics computation was evaluated by comparing the results provided by Analyzo and other existing tools such as CCCC[5], Cscope[6], Eclipse-Metrics[7], and Macxim/Spago4Q[8].

B. Sample and Data Collection

SourceForge.net shares its data to support Free Software researchers. In this study, we used the data available in a database managed by the University of Notre Dame[9] and another one provided by the FLOSSMole project[10]. We accessed these databases in November, 2009 and collected data about all the projects that matched the following criteria:

- *Source code written in the C language*. While the vast majority of Free Software applications is written in C [27], a large amount of research work focuses their analyzes in projects written in Java (e.g., the related work reported in section V). Given this disparity between the actual Free Software ecosystem and the research that addresses it, and our previous experience with analysis of Free Software written in C [28], we chose to focus the analysis in this work to such projects as well. This is our first study relating source code metrics and attractiveness, and in the future we plan to include other programming

[2]`softwarelivre.org/mezuro/analizo`

[3]`softwarelivre.org/mezuro/doxyparse`

[4]`doxygen.org`

[5]`cccc.sourceforge.net`

[6]`cscope.sourceforge.net`

[7]`metrics.sourceforge.net`

[8]`qualipso.dscpi.uninsubria.it/macxim`

[9]`nd.edu/~oss/Data/data.html`

[10]`flossmole.org`

TABLE I
DESCRIPTIVE STATISTICS

	Raw				Logarithm	
Metric	Minimum	Maximum	Mean	Std. Deviation	Mean	Std. Deviation
(Average) Coupling Between Objects	0.0015	711.50	2.26	9.04	0.35	0.98
(Average) Lack of Cohesion on Methods	0.0004	262.00	4.77	12.00	1.01	1.09
(Average) Structural Complexity	0	4,940.00	15.79	114.69	1.37	1.57
(Total) Number of Modules	1	7,177.00	74.98	276.54	3.08	1.39
(Total) Lines of Code	11	2,983,103.00	17,722.23	91,614.70	8.28	1.58
(Total) Number of Public Variables	1	516034.00	994.80	8850.44	4.91	1.80
(Total) Number of Functions	1	99468.00	612.54	2987.28	4.92	1.63
(Total) Number of Public Functions	1	99468.00	642.12	3025.94	5.02	1.58
(Total) Number of Members	1	288.00	2.90	6.19	0.59	0.79
(Total) Number of Downloads	6	941,498,760.00	956,674.26	17,760,732.37	8.20	2.66

languages (e.g., C++, Java), as well as trying to identify similarities and discrepancies among projects written in different languages.

- *More than one download.* Projects with no downloads are probably either non-development projects, or projects that have just started, or are other special cases.

This provided us with a list of 11,433 projects. After this preliminary sampling, the following steps were automatically executed by scripts developed by our group, to perform data collection:

1) Download the code of all the projects. This resulted in the source code for 10,128 projects since some of them had no available files (empty "files" section in the SourceForge.net project pages);
2) Run `Analizo` sequentially for all projects and store the computed metrics in a single database.
The metrics were successfully computed for 6,773 projects only, because (i) some downloaded files did not contain source code (e.g., binary-only downloads), (ii) the source code was not written in C, (the project was incorrectly classified as being written in C), or (iii) some files could not be processed by `Analizo` due to severe errors in the source code (e.g., syntax errors);
3) Cross-join the two datasets. Finally the two datasets – the SourceForge.net data available from the University of Notre Dame and FLOSSMole on the one side and the source code metrics calculated by `Analizo` on the other side – were cross-joined so that we could perform the needed statistical analysis.

Table I summarizes our sample, but the complete data set used for this study is available on the Web [11]. Section III-C discusses in detail how we selected the variables presented in Table I. This table shows natural values of minimum, maximum, arithmetic mean, and standard deviation for each variable, indicating the characteristics of our sample.

We analyzed our selected variables in their natural form (Raw) to verify their distribution, which is presented in the first part of Table I. Thereby, we observed that the Skewness and Kurtosis probability distribution showed high values, indicating non-normality [29]. Because of this non-normality, we transformed the variables to a logarithm scale for linearization, which reduced the Skewness and Kurtosis values and made them proper to run multiple regressions [20]. The arithmetic mean and standard deviation of the logarithm values can be seen in the second part of Table I.

[11] `ccsl.ime.usp.br/mangue/data`

C. Variables

Among the fourteen source code metrics that `Analizo` provided, we selected seven for our initial analysis: total LOC, total NM, total NOF, total NPV, total NPF, average LCOM4, and average CBO. To be able to apply to the procedural paradigm of the C language some metrics that area widely-used in the literature such as NOF, NPV, NPF, LCOM4, and CBO, we assumed a mapping of the object concepts of "class" and "method" to the C concepts of "source file" and "function".

In this first study, we limited the scope of metrics used to be able to reach a simple yet comprehensive model to relate source code and attractiveness. Thus, the ACC, AMZ_Size, COF, DIT, NOC, NOV, and RFC metrics were left out of the scope of this study.

Nevertheless, in the first stage of our statistical analysis, the LOC, NOF, NPV and NPF metrics showed a high correlation between each other, according to Pearson's parametric correlations as shown by the bold numbers in Table II. Highly correlated variables indicate that they are representations of a same attribute, making it unnecessary to use more than one. Since all these metrics represent a similar concept, we selected one of them – LOC – for our statistical analysis to reduce multicollinearity.

We have also analyzed the Spearman and Kendal non-parametric correlations (see Table III and Table IV, respectively) given that some of our variables are not normally distributed. In our analysis, we observed that after transforming our variables in their logarithmic form, Pearson correlations performed just as well as the non-parametric indices. Thus, we chose the Pearson parametric correlation because it represents the most commonly used form of correlation index, and it also provides the basis for the regression analysis we performed later. That way, we could maintain consistency once multiple regression techniques are based on parametric indices [29].

According to the analysis described above, we ended up considering LOC and NM as size metrics. Theoretically, the more LOC, the more NM. However, it is possible to have

TABLE II
PARAMETRIC CORRELATIONS: PEARSON

Variable	CBO	LCOM4	SC	NM	LOC	NPV	NOF	NPF	Mbrs	DLs
Coupling Between Objects	-	0.141	0.723	0.380	0.608	0.423	0.434	0.492	0.113	0.129
Lack of Cohesion on Methods	0.141	-	0.786	0.019	0.472	0.102	0.311	0.361	0.080	0.107
Structural Complexity	0.723	0.786	-	0.254	0.666	0.338	0.493	0.564	0.127	0.156
Number of Modules	0.308	0.019	0.254	-	0.799	0.730	0.815	0.827	0.311	0.344
Lines of Code	0.608	0.472	0.666	0.799	-	**0.872**	**0.923**	**0.927**	0.328	0.410
Number of Public Variables	0.423	0.102	0.338	0.730	**0.872**	-	0.756	0.761	0.303	0.386
Number of Functions	0.434	0.311	0.493	0.815	**0.923**	0.756	-	0.886	0.320	0.380
Number of Public Functions	0.492	0.361	0.564	0.827	**0.927**	0.761	0.886	-	0.308	0.365
Number of Members	0.113	0.080	0.127	0.311	0.328	0.303	0.320	0.308	-	0.676
Number of Downloads	0.129	0.107	0.156	0.344	0.410	0.386	0.380	0.365	0.676	-

TABLE III
NON-PARAMETRIC CORRELATIONS: SPEARMAN

Variable	CBO	LCOM4	SC	NM	LOC	NPV	NOF	NPF	Mbrs	DLs
Coupling Between Objects	-	0.340	0.803	0.473	0.662	0.523	0.518	0.566	0.156	0.169
Lack of Cohesion on Methods	0.340	-	0.773	0.213	0.490	0.348	0.478	0.516	0.129	0.162
Structural Complexity	0.803	0.773	-	0.370	0.685	0.478	0.571	0.631	0.164	0.196
Number of Modules	0.473	0.213	0.370	-	0.793	0.718	0.818	0.828	0.284	0.320
Lines of Code	0.662	0.490	0.685	0.793	-	**0.863**	**0.918**	**0.922**	0.307	0.392
Number of Public Variables	0.523	0.348	0.478	0.718	**0.863**	-	0.758	0.765	0.280	0.363
Number of Functions	0.518	0.478	0.571	0.818	**0.918**	0.758	-	0.895	0.300	0.362
Number of Public Functions	0.566	0.516	0.631	0.828	**0.922**	0.765	0.895	-	0.288	0.347
Number of Members	0.156	0.129	0.164	0.284	0.307	0.280	0.300	0.288	-	0.598
Number of Downloads	0.169	0.162	0.196	0.320	0.392	0.363	0.362	0.347	0.598	-

TABLE IV
NON-PARAMETRIC CORRELATIONS: KENDALL

Variable	CBO	LCOM4	SC	NM	LOC	NPV	NOF	NPF	Mbrs	DLs
Coupling Between Objects	-	0.244	0.650	0.341	0.483	0.377	0.373	0.410	0.118	0.114
Lack of Cohesion on Methods	0.244	-	0.597	0.148	0.341	0.240	0.333	0.362	0.097	0.109
Structural Complexity	0.650	0.597	-	0.262	0.497	0.345	0.413	0.460	0.124	0.132
Number of Modules	0.341	0.148	0.262	-	0.605	0.546	0.641	0.648	0.217	0.219
Lines of Code	0.483	0.341	0.497	0.605	-	**0.660**	**0.763**	**0.771**	0.233	0.270
Number of Public Variables	0.377	0.240	0.345	0.546	**0.660**	-	0.588	0.596	0.213	0.249
Number of Functions	0.373	0.333	0.413	0.641	**0.763**	0.588	-	0.864	0.228	0.248
Number of Public Functions	0.410	0.362	0.460	0.648	**0.771**	0.596	0.864	-	0.219	0.237
Number of Members	0.118	0.097	0.124	0.217	0.233	0.213	0.228	0.219	-	0.471
Number of Downloads	0.114	0.109	0.132	0.219	0.270	0.249	0.248	0.237	0.471	-

more lines of code without having more modules, by adding code into existing modules. Also, a software could have more modules but keep the number of lines of code when refactoring is applied. Furthermore, we understood that *number of modules* (NM) did not highly correlate with the others, since we considered a high correlation when the Pearson's correlations values were approximately 0.9 or higher as our criteria, emphasized in Table II.

In conclusion, LOC and NM were collected since they measure different kinds of size metrics, more or less influenced by programming languages and coding styles, respectively. Finally, to obtain the value of our structural complexity metric (SC), explained in Section II, CBO and LCOM4 were multiplied. These three metrics did not show high correlations with other metrics. As expected, SC showed a positive correlation with both. However, it was not as high as the others were, because CBO and LCOM4 had a low correlation with each other. This means that SC, statistically, represents different attributes when compared to CBO and LCOM4, thus endorsing the theory that CBO and LCOM4 together offer different information.

In summary, our multiple regression model ended up with the following variables:

- **Independent variables (source code metrics)**
  - *Structural Complexity (SC)*: The product of CBO and LCOM4 metrics.
  - *Lines of Code (LOC)*, the sum of lines of code in all modules of the project;
  - *Number of Modules (NM)*, the total number of all modules of the project.
- **Dependent variables (attractiveness)**
  - *Number of Downloads*: a proxy for the number of users of the project;
  - *Number of Members*: a proxy for the number of developers in the project.

The model developed in this study revolves around attractiveness, aiming at the explanation of its causes. We defined a multiple regression model that has attractiveness as its dependent variable. It was measured through two indicators: number of downloads and number of members. Thus, we have two different regressions, one for each attractiveness indicator.

They are the variables explained by the source code attributes proposed in our hypotheses. Consequently, the SC, LOC and NM metrics, which represent the source code attributes, are the independent variables – the influencers of attractiveness.

D. Research Hypotheses

In this first study about the relationships between source code metrics and attractiveness, we investigated whether two attributes – structural complexity and size – obtained via four source code metrics might influence the attractiveness of Free Software projects. Thereby, we can later observe whether these attributes influence people's perception of quality as consequence of attractiveness. According to the metrics chosen to represent structural complexity and size, we formulated three hypotheses:

H1 – *Free Software projects with higher structural complexity have lower attractiveness.* The higher the software complexity, the more difficult it is to understand its source code for maintenance and evolution purposes. This leads to an increase in the maintenance effort, and makes it more difficult to attract new members and users for the project. Over time, with less members and users, the project may lose its ability to add new features and fix bugs and, consequently, its ability to evolve and meet the user's changing requirements.

H2 – *Free Software projects with more lines of code have higher attractiveness.* To some extent, lines of code reflect the amount of features of the project and the amount of work that have been put into it. Therefore, projects with more lines of code will usually attract more users (since they have more features) and developers – since they offer more opportunities for contribution.

H3 – *Free Software projects with a higher number of modules have higher attractiveness.* The number of modules may indicate the project size and the possibility of working in parallel in independent modules. More modules may indicate a concern with good design and better modularization, which facilitates contributions. This attracts more members, who can write more features and fix more bugs, which would then attract more users.

IV. Hypotheses Testing

We specified a multiple regression model to explain the relationships between the selected source code metrics and attractiveness in Free Software projects. Before running this model, we analyzed and applied statistical techniques on the descriptive statistical values of our dataset presented in Table I, discussed in Section III-B. With the results in hand, we selected the variables of our regression model according to our scope definition, the analysis of the Pearson parametric correlations and Spearman and Kendal non-parametric correlations, shown in Table II, Table III, and Table IV respectively, and presented in detail in Section III-C. Finally, with the linearized values of SC, NM, and LOC (independent variables) and number of downloads and number of members (dependent variables), we tested our hypotheses according to our statistical multiple regression model compound for these variables.

Table V summarizes the regression results based on the Pearson's correlation values. These statistical results indicated a linear dependency between our source code metrics and each attractiveness variable. In this table, β is a coefficient that indicates the size of the influence of each metric on each attractiveness indicator.

As we can see in Table V, lines of code is more strongly correlated to downloads and members than structural complexity and number of modules, according to the standardized beta (*Std.* β). Standardized betas are calculated to perform comparisons between variables that are measured using different scales (e.g., lines of code and structural complexity). One cannot compare regular beta coefficients without first standardizing them.

Moreover, structural complexity has a negative correlation with attractiveness, as expected. Noteworthy is that the T-test and P (probability) values represent whether a source code metric is a statistically significant predictor or influencer of attractiveness indicators. For downloads, the number of modules is not significant because its P-value is greater than 0.05. Finally, in the last line of Table V, R-squared values indicate the percentage of attractiveness (users and developers) variance that this set of source code metrics is capable of explaining. So, roughly speaking, an R-squared of 20 percent indicates that a set of predictors can explain 20 percent of a dependent variable. We obtained the following equations:

$$\begin{aligned} downloads = \; & 1.551 - 0.286 \times log(SC) \\ & +0.856 \times log(LOC) + 0.008 \times log(NM) \end{aligned}$$

$$\begin{aligned} members = \; & -0.668 - 0.033 \times log(SC) \\ & +0.126 \times log(LOC) + 0.087 \times log(NM) \end{aligned}$$

Each equation has one R-value. The coefficient (β) of each variable is the size of influence that one of the source code metrics (the independent variable) has on the attractiveness – the dependent variable. So, one unit change in an independent variable generates a β-size influence on the dependent variable, on average.

The R-value represents the amount of the dependent variables that can be explained by that set of independent variables. In our analysis, the R-value indicated that source code metrics explain 18% ($R^2 = 0.180$) of the number of downloads and 12% ($R^2 = 0.121$) of the number of members. These are significant values for the social context that an adoption or volunteering of a Free Software projects are involved.

A. Hypothesis 1

The data analysis supports our first hypothesis – *Free Software projects with higher structural complexity have lower attractiveness*. In fact, structural complexity has a negative influence on attractiveness. When related to downloads, it presents a -0.286 β coefficient and $p < 0.001$. This means that structural complexity has an statistically significant impact on user interest.

In the Free Software context, structural complexity may indicate the difficulty to make improvements to the software, such as new features and bug fixes. So, most users may loose

TABLE V
EQUATIONS AND PEARSON CORRELATIONS

	Downloads				Members			
Metric	β	Std. β	T-value	P-value	β	Std. β	T-value	P-value
(Constant)	1.551	-	6.12	<0.001	-0.668	-	-8.47	<0.001
Structural Complexity (log)	-0.286	-0,150	-8.616	<0.001	-0.033	-0.058	-3.238	0.001
Lines of Code (log)	0.856	0.506	18.624	<0.001	0.126	0.249	8.846	<0.001
Number of Modules (log)	0.008	0.004	0.186	0.852	0.087	0.148	6.625	<0.001
R	0.425				0.348			
R^2	0.180				0.121			

interest in the software because another project may have a greater capacity to meet their evolving needs. Therefore, a smaller number of users, generating less reports could lead to less bug fixes and new features, which in turn could lead to less users in the future.

When related to members, structural complexity presents a β of -0.033 with $p = 0.001$, indicating that developers avoid to join projects with high structural complexity. A more complex source code is more difficult to understand and, consequently, to change. This may prevent new developers from joining the project. With fewer members, the community around a project is less active.

B. Hypothesis 2

The second hypothesis – *Free Software projects with more lines of code have higher attractiveness* – is also supported by our data. Lines of code has a positive influence on attractiveness.

For downloads, this metric has $\beta = 0.856$, with $p < 0.001$. In this context, lines of code can be an indication of the amount of software features and amount of work that have been put into the project so far. The more features available, the more users will become interested in the project. This may make the software more famous and more useful, attracting new members and users.

In addition, lines of code in relation to number of members indicated that developers are interested in larger projects. The β coefficient of this metric for members is 0.126 ($p < 0.001$). Therefore, for both downloads and members, lines of code is the metric with the highest influence because it is associated with software features and project size.

C. Hypothesis 3

The most interesting results were related to our third hypothesis – *Free Software projects with a higher number of modules have higher attractiveness*. For downloads, the data does not support the hypothesis: the high p-value ($p = 0.852$) does not allow us to claim that the number of modules has any influence on the number of downloads. For members, on the other hand, the hypothesis is confirmed: the number of modules influences the number of members with $\beta = 0.087$, and $p < 0.001$, which is statistically significant.

Both lines of code and number of modules are metrics that represent software size. The fact that both influence number of members, but only lines of codes influence the number of downloads makes us wonder whether they represent different characteristics of software size. In this context, lines of code probably is related to the amount of features in the project, which helps to attract both users and developers to the project.

However, when lines of code is kept constant, different values in the number of modules represent different ways of organizing these features over different modules. A higher number of modules thus indicates a higher modularity, which makes it easier for developers to work on the project and requires less coordination effort. For users, on the other hand, it is probably the case that it does not matter whether the software is modular or not; they are only interested in the provided features.

Finally, collaborators in Free Software projects often start participating in the project as users, attracted by the software features. After that, those users who have the potential to become developers may begin to contribute with the code. While a high number of lines of code (and thus of features) is enough to attract users, project leaders should pay attention to source code quality. To turn users into developers, the project has to provide a source code that is easy to understand and modify by keeping structural complexity as low as possible and modularity at a good level.

V. RELATED WORK

Large Free Software projects such as Debian GNU/Linux, GNOME, and KDE have invested in the creation of dedicated teams for quality assurance. These efforts involve everything from removing bugs and obsolete components to the definition of standards and strategies to prevent bugs and improve quality [5]. However, most projects do not have the resources to have a dedicated quality team.

Michlmayr *et al.* [5] performed a study on quality assurance problems in Free Software such as unsupported code, configuration management, security updates, users not knowing how to report bugs, the difficulty in attracting volunteers, lack of documentation, and problems with coordination and communication. None of these problems, however, are related to the quality of the source code *per se*.

Barkmann *et al.* [30] analyzed 146 Free Software projects written in Java, identifying the correlation between a set of object-oriented metrics and their theoretical ideal values. However, in their work the values of source code metrics were not associated with problems or attractiveness of Free Software projects.

Stamelos *et al.* [31] presented empirical results on the relationship between the size of application components and the delivered quality measured as user satisfaction. Quality characteristics of 100 applications written for GNU/Linux were compared to industrial standards. The results indicated that the so-called structural quality (e.g., component size) of an application is related to user satisfaction.

Midha [32] analyzed 450 projects from SourceForge.net and verified that high values of MacCabe's Cyclomatic Complexity and Haltead's Effort (complexity metrics) are positively correlated with the number of bugs and with the time needed to fix bugs. These metrics were also found to be negatively correlated with contributions from new developers, i.e., more complex code is less likely to attract new developers. However, Midha's study used complexity metrics measured at the subroutine level, while in our study we use complexity metrics at the module level.

Capra *et al.* [33] have shown that open governance is associated with higher software design quality on a study with 75 Free Software projects. They defined software design quality in terms of 5 Object-Oriented metrics, of which only CBO is used in our study. An open governance structure together with the lack of formal management and strict deadlines enables developers to enhance software design to have a high-quality product, since they do not suffer pressure to release the software [33]. Moreover, a better software design fosters a more open governance by allowing developers to work in independent modules without the need for explicit coordination activities. However, Capra's study has not addressed the issue of attractiveness.

Bargallo *et al.* [34] analyzed 56 Free Software projects, studying the relationship between software design quality and project success. They defined success in terms of downloads, page views and development activity, and design quality in terms of the object-oriented metrics CBO, DIT, MIF, and NOC. They found that the most successful projects exhibited lower design quality. They argue that perhaps in successful projects the main developers tend to shift their attention to lateral activities, such as replying to users in forums, instead of focusing on enhancing the code quality. Our results seemed to contradict theirs, but this is not the case. First, their conceptualization of success is different from our conceptualization of attractiveness. Moreover, we considered structural complexity in terms of CBO and LCOM4 metrics together, while they used a different set of metrics to represent the notion of design quality, having only CBO in common with the present study. Therefore, a straightforward comparison between their study and ours is not so simple.

VI. Conclusion

A systematic review of 63 empirical studies showed that there is little research addressing the characteristics or properties of Free Software projects, such as their quality, growth, and evolution [35]. Our study contributes with an unprecedented analysis of source code metrics from thousands of Free Software projects, causally linking software source code characteristics with attractiveness. In doing so, we expect to raise awareness on an important topic so far neglected. Free Software projects fail when they lack attractiveness. Therefore, understanding what influences attractiveness provides managerial knowledge to project leaders, pointing them to the right direction on prioritizing their resources.

Our results indicated that source code size and structural complexity explain a relevant percentage of the attractiveness of Free Software projects. Attractiveness is based on human perceptions and influenced by people's cognition, making it a complex issue, hard to understand and explain completely. Nevertheless, our study was able to explain 18% of software users and 12% of project developers, through a set of four source code metrics. These statistical results are significant for the social context that Free Software projects adoption and volunteering are inserted.

In this paper, we showed that lines of code (LOC) has a significant effect on the number of project users. Our results also indicated that structural complexity (SC) has a negative influence on project attractiveness. Therefore, a project will face greater difficulties to grow without observing some source code attributes such as cohesion, coupling, and modularity, which favors developer contributions such as new features and bug fixes.

In other words, our analysis indicated that software structural complexity growth may decrease the positive effects of new added features on attractiveness. Ideally, a project should keep its complexity constant as new code is incorporated, because developers are interested in improving the software, and the users in the improvements. This demonstrates to project leaders (in communities, foundations, governments, and companies) the importance to monitor metrics such as LCOM4 and CBO together with NM and LOC, thereby increasing their chances of forming a community of contributors around their software, further enhancing its quality. Thus, projects should grow managing their complexity, keeping the new members willingness to contribute.

Our study differs from related work because we analyzed a large sample of Free Software projects. Table I shows how diverse our sample of 6,773 projects is. There are projects with thousands of modules (7,177 – Broadcom replacement firmware[12]), millions of lines of code (2,983,103 – Broadcom replacement firmware), large structural complexity (4,940 – pyCDK[13]), several hundred members (288 – TinyOS[14]), and hundreds of millions of downloads (941,498,760 – MinGW: Minimalist GNU for Windows[15]). This sample was based on well-defined criteria and the number of projects involved provided us with statistical confidence in the results.

Nevertheless, this study has some limitations that motivate future work. Our sample is restricted to projects written in C available at SourceForge.net and our analysis to a limited set of metrics. In the future, we will include projects from other repositories, and extend this study to other source code metrics and programming languages such as C++ and Java.

[12]`sourceforge.net/projects/newbroadcom`
[13]`sourceforge.net/projects/pycdk`
[14]`sourceforge.net/projects/tinyos`
[15]`sourceforge.net/projects/mingw`

Furthermore, widely known projects such as GNU/Linux and Firefox should be included in studies of this kind, for their metrics may signal represent values that could be seen as targets or references.

Finally, we acknowledge that source code metrics are not the only variables capable of influencing attractiveness. Our previous work has identified that things such as the restrictiveness of the license, type of project, software life-cycle stage, and intended audience are all capable of influencing attractiveness [8]. At first sight, assuming that all these variables from our previous study are independent from the source code metrics we studied here, roughly 40% of attractiveness variance would be then explained. However, including variables in an equation in a statistically sound manner is not a trivial task. Accordingly, there is a need to further identify the interaction between that set of variables with the ones reported in this study.

Acknowledgments

The authors of this paper are supported by CNPQ, FAPESP, and the Qualipso project. This research has been developed in the USP Free Software Competence Center and the authors would like to thank Claudia Melo, Lucianna Almeida, Joenio Costa, Beraldo Leal, and Nelson Lago for their contributions.

References

[1] Y. Benkler, *The Wealth of Networks: How Social Production Transforms Markets And Freedom*. Yale University Press, 2006.

[2] A. Wasserman and E. Capra, "Evaluating Software Engineering Processes in Commercial and Community Open Source Projects," in *Workshop Emerging Trends in FLOSS Research and Development*, 2007.

[3] D. Riehle, "The Economic Motivation of Open Source Software: Stakeholder Perspectives," *IEEE Computer*, vol. 40, no. 4, pp. 25–32, 2007.

[4] Forrester-Consulting, "Open Source Paves the Way for the Next Generation of Enterprise IT," Forrester Research, Tech. Rep., 2008.

[5] M. Michlmayr, F. Hunt, and D. Probert, "Quality Practices and Problems in Free Software Projects," in *First International Conference on Open Source Systems*, M. Scotto and G. Succi, Eds., Genova, Italy, 2005, pp. 309–310.

[6] K. J. Stewart and S. Gosain, "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," *MIS Quarterly*, vol. 30, no. 2, pp. 291–314, June 2006.

[7] E. S. Raymond, *The Cathedral & the Bazaar*, T. O'Reilly, Ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999.

[8] C. Santos Jr., J. Pearson, and F. Kon, "Attractiveness of Free and Open Source Software Projects." in *Proceedings of the 18th European Conference on Information Systems (ECIS)*, Pretoria, South Africa, 2010, (forthcoming).

[9] E. E. Mills, "Software Metrics," Software Engineering Institute, SEI - Carnegie Mellon University, Tech. Rep., 1988.

[10] E. Tempero, "On Measuring Java Software," in *ACSC '08: Proceedings of the Thirty-First Australasian Conference On Computer Science*, vol. 74. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2008, pp. 7–7.

[11] T. C. Jones, *Applied Software Measurement: Assuring Productivity and Quality*. New York: McGraw-Hill, 1991.

[12] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*. Prentice Hall, 1994.

[13] K. Beck, *Smalltalk: best practice patterns*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[14] J. Bansiya and C. Davi, "Automated Metrics and Object-Oriented Development: Using QMOOD++ for Object-Oriented Metrics," *Dr. Dobb's Journal*, vol. 22, no. 12, pp. 42, 44–48, December 1997.

[15] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[16] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented Systems," in *Proceedings of International Symposium on Applied Corporate Computing*, 1995.

[17] D. P. Darcy, C. F. Kemerer, S. A. Slaughter, and J. E. Tomayko, "The Structural Complexity of Software: An Experimental Test," *Software Engineering, IEEE Transactions on*, vol. 31, no. 11, pp. 982–995, Nov. 2005.

[18] C. Richter, *Designing Flexible Object-Oriented Systems with UML*. Thousand Oaks, CA, USA: New Riders Publishing, 1999.

[19] J. P. Johnson, "Open source software: Private provision of a public good," *Journal of Economics and Management Strategy*, vol. 11, no. 4, pp. 637–662, 2002.

[20] K. Crowston and B. Scozzi, "Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development," in *IEE Proceedings Software*, vol. 149, no. 1, 2002, pp. 3–17.

[21] U. Raja and M. J. Tretter, "Investigating open source project success: A data mining approach to model formulation, validation and testing," Working Paper, Texas A&M University, College Station, Texas, Tech. Rep. Paper-071-31, 2006.

[22] M. Shaikh and T. Cornford, "Version management tools: Cvs to bk in the linux kernel," *Long Range Planning*, vol. 34, pp. 699–725, 2003.

[23] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: the apache server," in *ICSE '00: Proceedings of the 22nd international conference on Software engineering*. New York, NY, USA: ACM, 2000, pp. 263–272.

[24] V. Balijepally, R. K. Mahapatra, S. P. Nerur, and K. Price, "Are two heads better than one for software development? the productivity paradox of pair programming," *MIS Quarterly*, vol. 33, no. 1, pp. 91–118, 2009. [Online]. Available: http://aisel.aisnet.org/misq/vol33/iss1/7/

[25] K. Crowston, , and J. Howison, "Hierarchy and centralization in free and open source software team communications," *Knowledge Technology & Policy*, vol. 18, pp. 65–85, 2006.

[26] A. E. Hassan, Z. M. Jiang, and R. C. Holt, "Source versus object code extraction for recovering software architecture," *Reverse Engineering, Working Conference on*, vol. 0, pp. 67–76, 2005.

[27] G. Robles, J. M. Gonzalez-Barahona, M. Michlmayr, and J. J. Amor, "Mining Large Software Compilations over Time: Another Perspective of Software Evolution," in *Proceedings of the International Workshop on Mining Software Repositories (MSR 2006)*, Shanghai, China, 2006.

[28] A. Terceiro and C. Chavez, "Structural Complexity Evolution in Free Software Projects: A Case Study," in *QACOS-OSSPL 2009: Proceedings of the Joint Workshop on Quality and Architectural Concerns in Open Source Software (QACOS) and Open Source Software and Product Lines (OSSPL)*, M. Ali Babar, B. Lundell, and F. van der Linden, Eds., 2009.

[29] J. F. Hair, W. C. Black, B. J. Babin, R. E. Anderson, and R. L. Tatham, *Multivariate data analysis*, 6th ed. Upper Saddle River, NJ: Pearson Education In, 2006.

[30] H. Barkmann, R. Lincke, and W. Löwe, "Quantitative Evaluation of Software Quality Metrics in Open-Source Projects," in *AINA Workshops*, 2009, pp. 1067–1072.

[31] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code Quality Analysis in Open Source Software Development," *Information Systems Journal*, vol. 12, pp. 43–60, 2002.

[32] V. Midha, "Does Complexity Matter? The Impact of Change in Structural Complexity On Software Maintenance and New Developers' Contributions in Open Source Software," in *ICIS 2008 Proceedings*, 2008.

[33] E. Capra, C. Francalanci, and F. Merlo, "An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects," *IEEE Transactions on Software Engineering*, vol. 34, no. 6, pp. 765–782, Nov.-Dec. 2008.

[34] D. Barbagallo, C. Francalenei, and F. Merlo, "The Impact of Social Networking on Software Design Quality and Development Effort in Open Source Projects," in *ICIS 2008 Proceedings*, 2008. [Online]. Available: {http://aisel.aisnet.org/icis2008/201}

[35] K.-J. Stol, M. A. Babar, B. Russo, and B. Fitzgerald, "The Use of Empirical Methods in Open Source Software Research: Facts, Trends and Future Directions," in *FLOSS'09: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 19–24.