Study on the Social Impact on Software Architecture through Metrics of Modularity

Braulio Siebra[1], Eudisley Anjos[1,2], and Gabriel Rolim[1]

[1] CI, Centre for Informatics,
Federal University of Paraiba, Joao Pessoa – Brasil
eudisley@ci.ufpb.br,
{braulio.siebra,gabrielsrolim}@gmail.com
[2] CISUC, Centre for Informatics and Systems,
University of Coimbra, Portugal
eusis@dei.uc.pt

Abstract. Software systems have constantly increased in size and complexity. At the same time, software architecture also grows and becomes difficult to maintain leading to failures or abandonment of systems. According to Mirroring Hypothesis (MH), the organizational structure of the development team is a mirror of software architecture. So, the importance in understanding what changes in social structure can impact in the software architecture is crucial to avoid architectural problems. This work compares modularity metrics, applied to open-source systems, with the structure of developers inside the organization. The results show the relationship between the architecture and organization and contribute to guide the evolution and maintenance of systems.

Keywords: Modularity, Social Metrics, Software Architecture.

1 Introduction

The distributed software development has taking the attention of researchers since the 1990s [1]. This occurs because organizations have changed socially, economically and geographically, so they can distribute the resources aiming to increase productivity, improve quality and reduce costs in software development [2] . But, the physical distribution of teams amplifies existing problems in the system development process. Cultural differences, language, and other factors increase the difficulty in communication and coordination during the development process [3] , [4] .

When you have a large system, it is normal that appear several difficulties understanding and consequently maintaining it. The definition of " divide and conquer" used for algorithms can be used during the design process and reinforces the thought that the use of the concept of modularity offers quick results. Dividing a large problem into smaller problems, it requires less effort to find solutions. Thus, modularity can be understood as a way to facilitate maintenance.

The focus on dividing the system into modules and allow concurrent development has led researchers to compare the organization of the developers with the structure of

B. Murgante et al. (Eds.): ICCSA 2014, Part V, LNCS 8583, pp. 618–632, 2014.

the system modules, the system architecture. In their research they found that the structure of the product architecture was equal to the structure of the organization and called it Conways Law or Mirrorring Hypothesis [5], [6].

Although the structure of the organization has been taken into consideration in academia, many aspects inherent to the developer and the influence that the social environment can have over the system have still to be researched. These aspects can be named as: number of people who collaborate with the project , geographic location , number of commits , etc. , tell us social information that can be related to system modularity . In this work we use modularity metrics to understand the system architecture and compare them to information about the development team of open-source systems. We expect to obtain results that identify whether there is any relationship between modularity of a system and social aspects of the contributors. The results contribute to understand what can be done to improve the software evolution and maintainability.

2 Conceptual Background

In this section, we present the theoretical basis relevant to understand this work. The main topics are: software architecture, modularity, open-source software and mirrorring hypothesis.

2.1 Software Architecture

Software systems can increase in size and complexity, and when these systems grow it is important that the project has a good architecture. The problem of building software also involves decisions about the structures that form the system, the overall structure of control, communication, synchronization, and data access protocols, assignment of functionality to elements of the system, or on physical distribution of system elements [7]. These demonstrates the importance of understand the system architecture. Software architecture can play an important role in at least seven aspects of software development: Understanding Reuse, Construction, Development, Analysis, Management and Communication [8].

2.2 Modularity

According to [9], the concept of modularity emerged in the 1960s. Developed countries were facing a crisis due to the mismatch in the growth of hardware on the evolution of the software, the hardware in this crisis was developing rapidly while the technical software development progressed slowly. Since then, this concept has been growing and becoming essential in the development of a system.

In [10], several researchers have tried to define modularity. One of the best definitions was conceived by *Michael Jackson,* where he described modularity as a property of structuring software into modules, and module as an artifact designed to be constructed and understood separately.

Already, after *Parnas*, a module is independent, ie, you can change one without affecting the other. *Parnas* also says that a module can be developed concurrently, that is, one can develop while the other team is responsible. In [5], he stressed the idea of independence of modules in the sense of both cohesion and coupling, and created a concept about the apportionment of information that has become one of the most cited general conceptions in software literature. This concept is the "information hiding" or protection of information. He says that each module should keep to themselves the information that only he matters. To be applied to systems, this concept generates a weak coupling, because of this concealment of information.

2.3 Open Source Software

The term open source or open code was created by OSI (Open Source Initiative) and refers to free software. OSI is an organization dedicated to promoting open source software or free software organization. It was founded as a form of incentive for organizations to employ that concept. Its function is to check which licenses qualify as free software licenses, and are disseminating this concept showing the technological and economic advantages.

The OSI determines that the open source program should ensure free distribution, i.e., the license must not restrict the marketing or distribution of the program. The program must include source code and must allow distribution in compiled form.

2.4 Conway's Law and Mirrorring Hypothesis

The concept of Conway's Law came in 1968 when the scientist and programmer Melvin Conway said any organization that makes system design will produce a design whose structure is a copy of the structure of the enterprise architecture [11]. That means organizations that want to design systems or generate products that copy the structure of how the company is organized, i.e. its architecture. It appears these days with a new terminology known as Mirroring Hypothesis.

The Mirroring Hypothesis allows the relationship between technological modularity and organizational structure. Fewer lines of code can result in a faster time with fewer defects. Therefore, the reuse is important not only to write less code, but also because it means find and fix problems quickly and at no charge.

There are other works approaching the relation between modularity and open-source systems. In [12], for example, was made a survey showing that the open-source systems were less coupled than proprietary systems. Besides prefer modular systems, open source communities also invest in tools and social practices based on openness and transparency technique.

3 Metrics of Modularity

According to [13], software metrics are designed to identify, measure and allow controlling the main parameters that affect software development. The software

metrics used in this work are described below. It is important to mention that all metrics were selected to be applied to open-source object-oriented systems.

- **TLOC – Total Lines of Code:** The count of lines of code is generally used as a benchmark for other metrics. As the name implies, this metric will count the number of rows in the code.
- **NOPK - Number of Packages:** Number of packages is defined as the count of the packages in the selected scope for analysis. The metric includes all sub packages.
- **NBD - Nested Block Depth:** This metric represents the maximum number of blocks of code nested in a particular method of a class.
- **CC – Cyclomatic Complexity:** Developed by Thomas J. McCabe [14], the CC measures the level of complexity of a method or function by counting the paths in code with independent execution. This metric is largely used in academia and industry and one of the most important to us.
- **LCOM – Lack of Cohesion of Methods:** According to [15], cohesion is a qualitative indication of the degree to which a module focuses on just one thing. This metric measures how a class is not cohesive. I.e., the higher the number of LCOM the less cohesive is the class.
- **Ca - Afferent Coupling:** The measure proposed by afferent coupling in [16] is the number of different classes that relate to the current class by means of fields or parameters
- **Ce - Efferent Coupling:** Opposed to the afferent coupling, [16] also proposed the efferent coupling metric which is the number of different classes that the current class references through fields or parameters.
- **WMC - Weighted Methods per Class:** According to [17], this metric measures the individual complexity of a single class. The number of methods of a class and its complexities are indicators that time and effort are required to the development and maintenance of classes.

4 Methodology

In this section we describe: the open-source systems used as case study, the tools and methods used to collect the values of the metrics for software modularity and social analysis, and how the tests were performed.

4.1 Open Source Systems

The open-source systems provide their code openly and work with distributed development; therefore it is possible that contributor's social and cultural aspects influence the project modularity. For this reason and the fact that in open-source systems the source code is at our disposal, these systems have been our focus for testing.

It was necessary to compile the code for each project to obtain the numerical values of each metric. Eight systems written in Java were selected for case study:

FindBugs, HyperSQL Database Engine, JasperReports, jEdit, JMeter, Poi, TomCat and Vuze.

We point out that the systems choice mentioned above was also a consequence of their presence in GitHub repository software. Projects that were not hosted on GitHub were excluded from the scope of our research.

A version of each of the selected systems was chosen for testing. Each system and its selected version are shown in Table 1:

Table 1. Description and version of the systems used as study cases

Open Source Systems	Description	Version
FindBugs	Uses static analysis to look for bugs in Java code.	**3.0.0**
HyperSQL Database Engine	It is a basic server written entirely in the Java language data.	**2.3.2**
JasperReports Library	It is a reporting tool that produces documents that can be viewed, printed or exported to various formats (PDF, HTML, XML, etc.)	**5.5.2**
JEdit	It is a text editor developed in Java.	**1.6**
JMeter	It is a tool used for load testing services offered by computer systems.	**2.12**
Poi	Provides pure Java libraries for reading and writing files in Microsoft Office formats such as Word, PowerPoint and Excel.	**3.11**
Tomcat	It is a Java web server. The Tomcat is a JEE application server, but it is not a server EJBs.	**8.0**
Vuze	It is a Java program that allows downloading files via the BitTorrent protocol	**5.3.0.0**

4.2 Git / GitHub

Git is a version control system widely used by software development companies. Git helps large contributors teams to keep the systems organized and documented. Anything implemented in software or commented can be found anywhere at any time.

GitHub is a web hosting service designed for distributed projects that uses the Git version control. Thus, it is used as an online repository of source code for open source projects. Information about all commits (updates) of projects can be found there. It uses a social network that allows people to follow the project development; a feature to view graphs with the amount of updates per contributor, and other features. The eight systems used in the case study are hosted by GitHub.

4.3 Choice of Tools

After several readings and studies Eclipse Plugin 1.3.6 Metrics tool was chosen. This decision was based on the fact that all the chosen systems were written in Java and needed to be compiled to generate the metrics. This tool collected various metrics related to project modularity. Eight metrics that can influence systems modularity have been selected after reading some articles.

The research to find mining tools repository was done cautiously. The selected tool - GitStats- generates statistics of any project that is hosted on Git and outputs data such as: quantity of contributors, most active contributors, amount of commits, etc... The choice for GitStats happened for many reasons. Primarily because it accesses the Git server, where the eight selected systems are hosted. Furthermore, we assume the GitHub repositories as one of the best nowadays.

Both the Eclipse Metrics Plugin 1.3.6 and GitStats- are easily found tools used by businesses. The use of such tools allows us to perform a job that can be easily replicated in an inexpensively way by companies- not limiting this research to academic context.

4.3.1 Metrics Collected via Eclipse Metrics Plugin 1.3.6

In collecting data 23 metrics were found. Here, we present the eight metrics used in our research. There were two size metrics, three complexity metrics, two coupling metrics and one cohesion metric. The classification of each used metric is shown below in Table 2.

4.3.2 Metrics Collected by GitStats

The GitStats was used to generate some relevant social characteristics to come across the software metrics. This tool provided the quantity of contributors present in the GitHub repository of each project, the amount of commits by author and the total amount of commits for each system.

An essential feature that is not presented by GitStats is geographic location of each contributor. The location is very important to understand how modular the organization we are evaluating is. This idea was presented before by [12] and helped us to compare the organization structure with the source code structure. To obtain the contributors' location we executed solicitations using GitHub API to get the e-mail address and name. Through this information we could find some contributors' locations.

Table 2. Classification of metrics

Metrics	Size	Complexity	Cohesion	Coupling
TLOC	**X**			
NPOK	**X**			
NBD		**X**		
CC		**X**		
WMC		**X**		
LCOM			**X**	
Ca				**X**
Ce				**X**

We assumed that only the 20 most active contributors of each project were relevant. The amount of commits guided this assumption. We noticed that beyond 20, the amount of commits became irrelevant for our purposes . It is important to mention here that it was not possible to find the location of all twenty contributors in each system. The difficulty was due to lack of information from the GitHub on the contributors. Sometimes just a nickname or a duplicated name was provided.

Thus, the geographical distribution is given as shown in (1) and the results of the related metrics are shown in the Table 5:

$$\text{Number of Countries / Number of Contributors} = D_G \,. \tag{1}$$

5 Results

The first results were obtained using the tools mentioned in the section 4.3: Eclipse Metrics 1.3.6 and GitStats. Eclipse Metrics calculated metrics for each of the eight projects. We selected the most relevant metrics for this paper and the results are shown in Table 3 and Table 4.

The GitStats was used to generate some relevant social features to compare with the software metrics. Among the many metrics, this tool showed we selected the three most important ones : the quantity of contributors present in the GitHub repository of each project, the amount of commits of each author and the total amount of commits for each system.

Table 3. Size and complexity using *Metrics for Eclipse*

Open-Source Systems	TLOC	NPOK	NBD	CC	WMC
FindBugs	20.573	77	1,379	2,678	26.410
HyperSQL	69.191	32	1,781	3,569	36.824
JasperReports	235.798	116	1,460	2,045	37.726
JEdit	117.366	42	1,590	3,054	22.347
JMeter	32.764	43	1,485	1,879	5.882
Poi	86.762	48	1,317	1,863	17.148
Tomcat	150.834	102	1,553	2,611	33.013
Vuze	565.168	488	1,851	2,713	85.275

Table 4. Coupling and cohesion using *Metrics for Eclipse*

Open Source Systems	LCOM	Ca	Ce
FindBugs	0,267	25,494	11,325
HyperSQL	0,366	35,625	12,00
JasperReports	0,247	40,397	17,129
JEdit	0,222	18,929	8,429
JMeter	0,221	16,256	6,465
Poi	0,201	26,917	14,562
Tomcat	0,289	17,804	5,667
Vuze	0,318	22,795	4,814

5.1 Results Analysis

Here we present a comparison between modularity and social metrics. To evaluate the results we formulated five hypotheses according to [6] [17], [18]. We divided into subsections for more relevant comparisons and discuss each of the results.

Table 5. Result of social metrics

Open Source Systems	Contributors	Commits	D_G
FindBugs	26	14.743	0,36
HyperSQL	12	5.361	0,44
JasperReports	20	6.961	0,37
JEdit	37	6.581	0,58
JMeter	29	10.332	0,55
Poi	33	5.483	0,57
Tomcat	24	12.067	0,60
Vuze	42	24.828	0,54

Table 6. More active author of each project

Open Source Systems	Author/Commits (%)
FindBugs	Bill Pugh (EUA) / 50,04%
HyperSQL	Fredt (ING) / 46,32%
JasperReports	Teodord (ROM) / 48,36%
JEdit	Slava Pestov (RUS) / 33,33%
JMeter	Sebastian Bazley (ING) / 66,72%
Poi	Nick Burch (ING) / 29.45%
Tomcat	Mark Thomas (ING) / 66,81%
Vuze	Parg (EUA) / 27,96%

5.1.1 Correlation between Coupling and Geographical Distribution

A análise entre a distribuição geográfica e o acoplamento foi direcionada a partir da hipótese que segue:

H_1: to maintain system modularity an increase in geographical distribution implies coupling reduction.

This happens because once they have contributors in different countries, the system must be least coupled to facilitate development.

In order to see better what happens, we analyze in particular the result of metrics for two systems: Tomcat and JMeter. As shown in the previous tables, the most active contributor in each system was responsible for over 66% of the total commits . This is an indication that the code should be more coupled because a single person dominates and understands more the code. But when we analyze the coupling metric (Ca, Ce) we realized that Tomcat and JMeter are among the least coupled.

Although there is a contributor who dominates the code more than the others , those systems are among the most widely distributed geographically, as shown in Table 5. With greater geographical distribution, the code tends to become least coupled . A weak coupling indicates less dependence between modules, i.e., there is a tendency for systems to be more modular. One of the issues raised here for future investigation is which attribute is more powerful than the other.

5.1.2 Correlation between Quantity of Contributors and Complexity

This analysis was based on the following hypothesis:

H_2: to maintain system modularity the growth of contributors results in a decrease in system complexity.

This happens because once there are many developers , the system needs to be less complex to facilitate development.

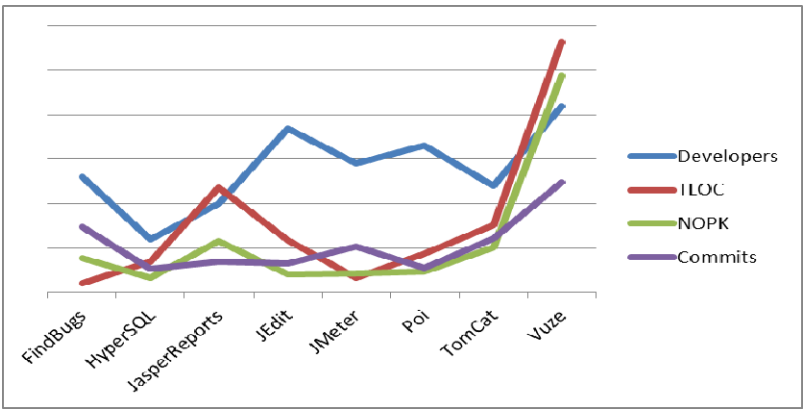

Fig. 1. Overview of each system

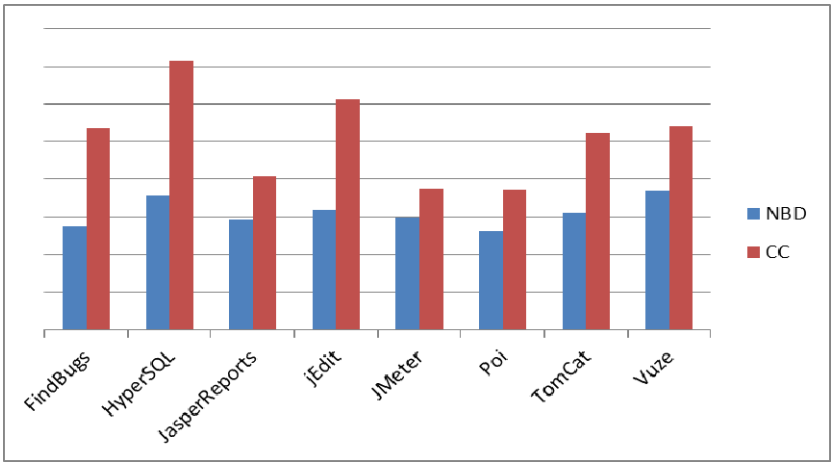

Fig. 2. Complexity of systems

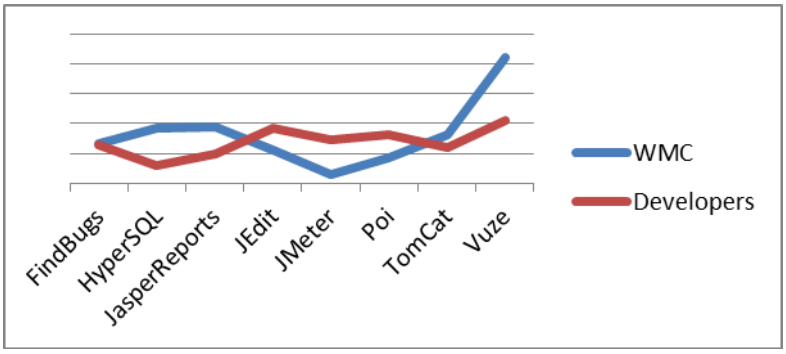

Fig. 3. Correlation of number of developers with the WMC

In order to understand better what happens, we analyze especially the performance of Vuze system. In Table 6 we see that the contributor with a larger quantity of commits in Vuze holds only 27.96% of commits . Thus, Vuze is better split among contributors. In addition, of all projects analyzed, it is the one with the largest quantity of contributors. By having this feature, its complexity should tend to a lower value. But to collect the results of the metrics NBD and DC detected that Vuze is a complex system, as shown in Fig 2. In NBD metric it was the one with greater complexity, in DC metric it was the third most complex.

The WMC metric returns a value of overall complexity, while the CC and NBD calculate only the average complexities. For this reason we thought the use of CC and NBD metrics would not be sufficient and decided to use the WMC metric to assess the social evolution of architecture. Hence, we would evaluate the system version altogether, so it could consider the quantity of contributors. But according to Figure 3, we concluded that Vuze is still the most complex even when we analyze the system complete version.

A possible explanation for Vuze to be among the most complex systems can be seen when observing Figure 1, which shows Vuze as the system with the largest quantity of contributors, more lines of code, the largest amount of packages and larger quantity of commits of all the eight systems investigated.

5.1.3 Correlation between Cohesion and Geographical Distribution

In the analysis of the geographical distribution and LCOM metric we used as a starting point the following hypothesis:

***H*$_3$**: *to maintain system modularity, high geographic distribution implies in greater cohesion.*

This happens because once they have contributors in different countries; the system must be more cohesive in order to facilitate development.

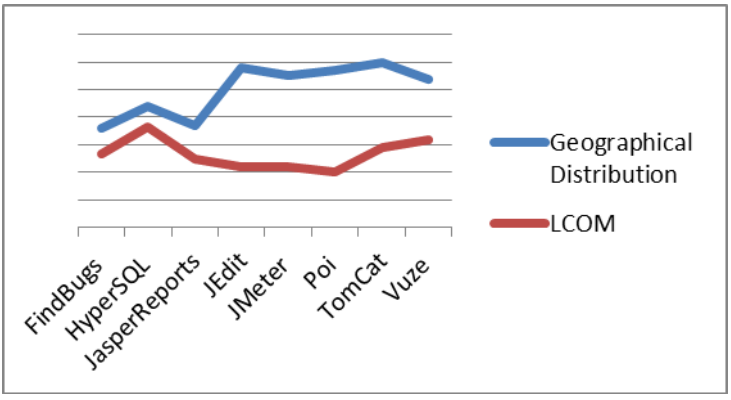

Fig. 4. Correlation of geographical distribution with cohesion

In Figure 4 we can see that jEdit, Poi and JMeter systems are among the systems which are geographically dispersed and are among the most cohesive. Moreover, Vuze and HyperSQL and FindBugs are among the least geographically dispersed and are those with low cohesion. However, the most surprising result was Tomcat, it was the system most widely dispersed and unexpectedly was among the least cohesive. The study of the unexpected behavior of this system is currently being developed by our researchers.

5.1.4 Correlation between the Quantity of Contributors and the Amount of Packages

This analysis was based on the following hypothesis:

***H_4**: to maintain the system modularity the growth in t quantity of contributors implies in a growth in the amount of architectural components.*

This is because as the quantity of contributors increases, the system tends to increase the amount of packages.

Due to the data we collected from metrics, we considered the architectural components as packages. In order to see better what happens, we analyze in particular the behavior of Vuze and HyperSQL systems. Vuze is the one with more people contributing and thus more packages. HyperSQL is the one with fewer contributors and less packages. These two systems follow the trend we assumed in our hypothesis. However, as some of the systems do not follow this trend, we cannot conclude the hypothesis is met in all cases. To better understand this, we must know how close these people are in software design to understand cohesion, perhaps a social network analysis could give us more information on this comparison. Or even we could examine the architectural components considering another scope, larger or smaller than package.

5.1.5 Correlation between Geographical Distribution and System Complexity

The values obtained were quite motivating . Whereas a modular system has a low complexity, the hypothesis initially raised for this comparison was as follows:

***H_5**: the best geographically distributed a system is, the least complex.*

This is because once they have contributors in different countries; the system needs to be less complex to facilitate development.

We can note from Figure 5 that when geographical distribution increases, there is in most cases, a reduction in system complexity. Some systems might not follow the trend, although this issue can be further tested by making a study of the evolution of the system and proving that for a particular system, when the geographical distribution increases, the complexity decreases. These tests are being conducted by our team for future work.

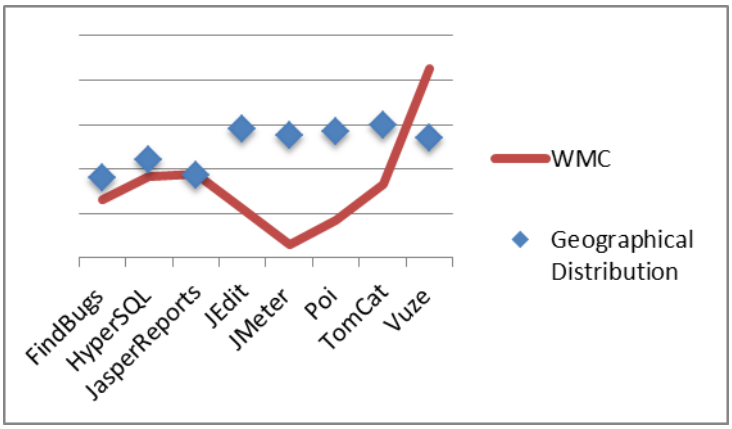

Fig. 5. Geographic distribution graph relating to Complexity

6 Conclusion and Future Works

Among the 8 open source systems investigated, there are few who escape completely the standard required to understand their behavior before the results of the metrics of modularity. They are:

- JasperReports: It is in 6th place in the complexity, i.e., it has low complexity and with only 20 people, it is the 2nd project with fewer contributors. Its geographical distribution is the 2nd smallest, which does not explain the low complexity. Therefore, we could not correlate the social aspects of modularity metrics in this system.
- JEdit: It is 2nd largest system in quantity of contributors and the 2nd largest in geographical distribution. The complexity of this system is the 2nd largest of all. To evaluate the system considering its modularity, system complexity should have been lower.
- Poi: It has large number of contributors, but the system complexity is low. Several people developing the project in many different countries makes the code less complex since it is highly splitted.
- JMeter: It is among the projects with the largest number of contributors, there are 29. It is among the largest geographic distributions. Because of this, its complexity is the 2nd lowest.

These results imply that we need to research more to understand why these systems are exceptions. Maybe calculating more metrics could let us see new features. The other results were more encouraging, proving that there is some relationship between social metrics and modularity in the development of a system.

The research confirms the initial idea that social parameters for system projects can be related to modularity metrics. However we still have difficulties on how to obtain social data for analysis. Although we have seen here and in other researches is that,

according to the Mirroring Hypothesis, the organization mirrors the code, nevertheless working with open-source systems is a bit complicated. There are many peculiarities to consider with those systems, thus a great amount of metrics has to be analyzed to obtain a better result.

The results evidence the necessity of trade-off analysis among the metrics. This will underpin the peculiarities of each metric. Besides, to improve results it is necessary to increase the quantity of versions and the amount of metrics for each system. These tests are being conducted by our team.

We also intend to restrict the term "Social Factors", because it is far reaching. We will try to analyze other social factors associated to the contributors to get more responses. These social factors would be: education, age, occupation of the contributors among others.

We will investigate the subcomponents of the system package structure. It is usually just a physical hierarchy, but logically there are other components, such as different subprojects, systems and libraries all of them maintained by different sub teams of contributors.

We will also check possible explanation on comparing and collecting every threat that affect the results. Every threat will be analyzed in order to ascertain its influence and its validity concerning the research.

References

1. Carmel, E., Tija, P.: Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce. Cambridge University Press, New York (2005)
2. Audy, J., Prikladnicki, R.: Desenvolvimento Distribuído de Software:Desenvolvimento de software com equipes distribuídas. Elsevier, Rio de Janeiro (2008)
3. Lanubile, F., Damian, D., Oppenheimer, H.: Global Software Development: Technical, Organizational, and Social Challenges. ACM SIGSOFT Software Engineering Notes 28(6) (November 2003)
4. Pilatti, L., Audy, J.L.N., Prikladnicki, R.: Software configuration management over a global software development environment: lessons learned from a case study. In: Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner (GSD 2006), pp. 45–50. ACM, New York (2006)
5. Morris, R., Parnas, D.L.: On the Criteria To Be Used in Decomposing Systems into Modules. Magazine Communications of the ACM (1972)
6. Baldwin, C.Y.: Modularity and Organizations, Harvard Business School Finance Working Paper No. 13-046 (2012), http://ssrn.com/abstract=2178640 or http://dx.doi.org/10.2139/ssrn.2178640
7. Garlan, D., Shaw, M.: An introduction to software architecture. Advances in software engineering and knowledge engineering 1, 1–40 (1993)
8. Garlan, D.: Software Architecture. School of Computer Science at Research Showcase (2001)
9. Rezende, D.A.: Engenharia de Software e sistemas de informação. 2nd ed. Brasport, Rio de Janeiro (2002)
10. Gabriel, R.P., Jackson, M.: Definitions of Modularity. Retrospective on Modularity. AOSD, Porto de Galinhas, Brazil (2011)

11. Conway, M.E.: How Do Committees Invent? Magazine Datamation (1968)
12. Maccormack, A., Rusnak, J., Baldwin, C.: Exploring the Duality between Product and Organizational Architectures: A Test of the "Mirroring" Hypothesis. Publication in Research Policy (2011)
13. Mills, E.E.: Software Metrics (CMU/SEI-88-CM-012). Software Engineering Institute, Carnegie Mellon University (1988)
14. McCabe, J.: A Complexity Measure. In: Proceedings of the 2nd international conference on Software engineering (ICSE 1976), p. 407. IEEE Computer Society Press, Los Alamitos (1976)
15. Pressman, R.S.: Engenharia de Software, 5th edn., p. 843. McGraw-Hill, Rio de Janeiro (2002)
16. Martin, R.C.: Prentice-Hall, Inc., Upper Saddle River (1995)
17. Rosemberg, L.H.: Applying and Interpreting Object Oriented Metrics. [S.l.]: NASA Software Assurance Technology Center, SACT (2007)
18. Cai, Y., Huynh, S.: Measuring Software Design Modularity, pp. 5–6 (2008)