Procedural x OO
A Corporative Experiment on Source Code Clone Mining

José Jorge Barreto Torres[1], Methanias C. R. Junior[1] and Mário André de Freitas Farias[2]
[1]*Federal University of Sergipe – UFS, Sao Cristovao, Sergipe, Brazil*
[2]*Federal Institute of Sergipe – IFS, Aracaju, Sergipe, Brazil*

Keywords: Software, Mining Software Repositories, Clones, Experimental Software Engineering, Closed-source Projects.

Abstract: Open Source Software (OSS) repositories are widely used to execute studies around code clone detection, mostly inside the public scenario. However, corporative code Repositories have their content restricted and protected from access by developers who are not part of the company. Besides, there are a lot of questions regarding paradigm efficiency and its relation to clone manifestation. This article presents an experiment performed on systems developed in a large private education company, to observe and compare the incidence of cloned code between Object Oriented and Procedural proprietary software, using an exact similarity threshold. The results indicate that Object Oriented Software wondrously showed higher cloned lines of code incidence and a similar use of abstraction (clone sets) for functions or methods.

1 INTRODUCTION

The demand for speeding up software development allied to the lack of patterns and the inexistence of internal policies to implement best practices triggers a series of issues related to coding organization. Software development teams have to achieve business deadlines, so they adopt the bad practice to copy-and-paste code. In this way, clones populate software repositories and hinder the improvement or maintenance of systems.

There are some reasons for the existence of clones: The most part of legacy systems code is the result of reusing existing code, so, developers who want to implement a new feature find some code snippet similar to the desired one then make a copy and modify it; Some code fragments used on default messages are copied to maintain a standard coding style, also generating clone code; Similar computing instances or code that perform similar computing are often cloned, even without the act of copy-and-paste, because the operations are similar; Some clones result from identical instructions that works only with different data types – this indicate the failure to use Abstract Data Types; Systems that have time constraints and need frequent optimization updates to computing replications, especially when the compiler does not provide inline expressions insertion; Occasional code fragments that are accidentally identical – as applications increase in size this type of accident occurs more often (Baxter et al., 1998).

Construction of device drivers also generates many similarities between codes, as much of this type of program geared to the same platform is virtually identical, only having some attributes and parameters modified (Ma and Woo, 2006). Moreover, another reason for the existence of clones is called “reinventing the wheel”, because some developers do not bother to look if there is already a piece of code for something that was requested to the team (Marcus and Maletic, 2001).

Despite the copy-and-paste way be more productive, this attitude may cause a serious maintenance problem, for example, in case of bugs. If a bug was found in a piece of code that has been cloned in several other pieces, all of these clones should be corrected so that the bug is completely resolved (Khatoon, Mahmood and Li, 2011).

Most studies around clone code theme make use of the same concepts. For example, the main types of code clones are (Rattan, Bhatia, and Singh, 2013): Exact clones or program fragments identical to each other; Parameterized clones, are fragments with the same structure except for changes in data types, identifiers, layout and comments; Near-miss clones,

Torres, J., Junior, M. and Farias, M.
Procedural x OO - A Corporative Experiment on Source Code Clone Mining.
DOI: 10.5220/0006325003950402
In *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017) - Volume 2*, pages 395-402
ISBN: 978-989-758-248-6

program fragments copied with a few modifications inside; Semantic clones, blocks of code textually different but producing a same computation.

Other authors bring some terminologies concerning the relationships between clones (Roy and Cordy, 2007). A Clone Pair is a pair of code fragments identical or similar to each other. To illustrate, we can turn attention to Figure 1 and note that we have three code fragments which we will name in a short way as F1, F2 and F3. From these three fragments we can mount five clone pairs: <F1(a),F2(a)>, <F1(b),F2(b)>, <F2(b),F3(a)>, <F2(c),F3(b)> and finally <F1(b),F3(a)>. A Clone Class is a set of code fragments in which any two of the members can form a clone pair. In short, a clone class is the union of all clone pairs who shares code fragments in common. Clone Family, also known as Super Clone, is the group of all clone classes belonging to the same domain.

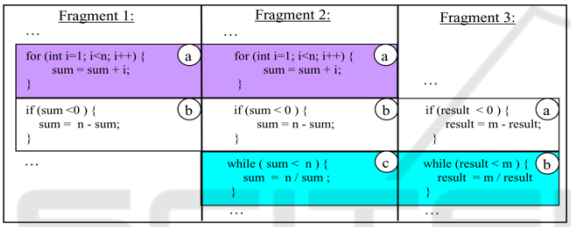

Figure 1: Examples of clone pair and class (Roy and Cordy, 2007).

Despite code clones are considered harmful (Kapser and Godfrey, 2008), for all the reasons we presented earlier, in some cases they may be a good choice. Introducing a new feature inside existing software can be eased by replicating the code and making the modifications. When the modified version of the code is tested in a sandbox or something similar, it can be applied in the production environment. This way minimizes the risk of instabilities in the stable version.

Some studies suggest that code clones may be avoided by adopting good design techniques and development methodologies, including refactoring on the development process (Roy and Cordy, 2007). Many efforts show that code refactoring as part of the package of a clone detection tool may be a desirable feature in some situations. Roy and Cordy (2008) studied cloning incidence in both C and Java Open Source Systems, executing a mixed experiment with different paradigms, showing interesting results regarding Clone Classes and Clone Sets incidence.

An Open Source System is publicly accessible and people can modify and share it. The software where only one person, team or organization who created it has access to modifications is called proprietary or closed source software (OpenSource.com).

Thus, considering a corporate environment with a well-defined software process, this paper aims address the following research question: “Have object-oriented software systems more efficiency than procedural systems, regarding code clone manifestation?” The question is about a proposal of efficiency in OO coding regarding Procedural, due to present abstraction structures in the Object-Oriented paradigm. The utilization of those abstraction structures are intended to provide a better code reuse and consequently less clone manifestation. To answer, our experimental evaluation analyzed large-scale Closed-Source Systems and compared their OO Systems with the Procedural ones. This is an in-vivo evaluation and the results are generalizable evidence only for similar teams, projects and environments. Despite OO languages are intended to have a better abstraction implementation, in our industrial environment, Procedural and OO systems presented similar behavior regarding clone manifestation. Those results showed numbers that leave opened other issues who are not directly linked to the paradigm question.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 is dedicated to the understanding of the main tool used in our experiment. Section 4 presents the experiment planning and definition. Section 5 describes the experiment execution among with the environment used to explore the clone code detection. Section 6 describes, analyzes and discusses the validity of the obtained results. Finally, Section 7 contains conclusions and final remarks.

2 RELATED WORKS

As this work focuses on clone incidence inside software repositories, this section presents studies about this subject. The main peculiarity of these articles regarding our work is that they were performed inside Open Source Software (OSS) environments.

Roy and Cordy (2011) developed a qualitative evaluation along with a comparison of techniques and clone detection tools. In their work, some key concepts also have been described with a generic clone detection process and taxonomy. They used a hybrid clone detection tool called NICAD to

examine more than 15 open source C and Java systems.

The same researchers mentioned above provided in another work (Kim, Sazawal, and Notkin, 2005) a description of commonly used terms, review of existing clone taxonomies, detection approaches and experimental evaluations of clone detection tools. At last, a list of some problems related to clone detection for future research is presented and discussed.

Many works were concerned about evaluating source code mining techniques and tools, identifying their strengths and weakness. Khatoon, Mahmood and Li (2011) try to extract positive and negative aspects from cloned detection tools and techniques to help future researchers and developers.

Schwarz, Lungu and Robbes (2012) focused on large code bases, combining three lightweight clone detection techniques to evaluate performance on a real-world ecosystem. The techniques are directed to three types of clones. The type 1 are Hashes of Source Code. Type 2 are defined as Hashes of Source Code With Renames. A clone is considered a type-2 if it is a type-1 even after every sequence of alphabetical letter be replaced by the letter "t" and all sequence of digits replaced by number 1. Type 3 or "Shingles", are defined as a consecutive sequence of tokens in a document, after the transformations defined by rules of type-2 clones.

Roy and Cordy (2008) motivate our work. Their study was about finding function clones inside C and Java Open Source Code repositories, with projects varying in size from 4K LOC to 6265K LOC. All non-empty functions with a minimum of 3 LOC were considered, that includes the function header with opening and ending bracket and at least one code line. The validation of results was done by hand and using Linux diff tool to check the textual similarities. Like them, we run a clone detection tool in two different software repositories with the difference that they are protected repositories belonging to a private corporation. We intend to compare incidences of cloned code between Object-Oriented and Procedural Projects.

Clone Mining research needs substantial infrastructure support, particularly with respect to adopting a standard experimental process, described in some Mining Software papers (Colaço et al., 2012) and in this paper, with the goal of effectively replicating clone studies. The barrier and cost for experimentation with Clones Mining are considerably low compared to other software engineering techniques (e.g., on-line experiments with participants). In other words, research projects and papers can conceive an experience factory and demonstrate true value of this area for practitioners.

3 CloneDR TOOL

CloneDR uses a tree-based technique called Abstract Syntax Trees (AST). Supporting a variety of language dialects and the capacity of huge sets of files analysis, this tool is top-rated in literature (Roy, Cordy, and Koschke, 2009) with a more sophisticated detection of clones. The intention for choosing this tool was to maintain the same type of analysis and results pattern of the referenced work. Also, the tool was the only one available to our team, for Java and PL/SQL code analysis.

The AST technique consists in receiving tree-parsed code fragments to find exact clones by hashing the sub-trees and comparing them. To locate near-miss clones, a bad-hashing function is used to preserve the main properties of this type of clone. For example, this function may ignore only identifier names, building a hash code for the rest.

A better description of this technique is presented by Baxter et al. (1998). At first, all the program code is fragmented in parts that will be compared to find out which one are equivalent. After this parsing stage, an Abstract Syntax Tree is build and some algorithms are applied to find clones. The first one is called the Basic algorithm and it is responsible to detect sub-tree clones. The second one, called the sequence algorithm tries to detect variable-size sequences of sub-tree clones and it is used essentially to detect statement and declaration sequence clones. The third algorithm attempts to find more complex near-miss clones, generalizing combinations of other clones. AST technique does not concern to detect semantic clones. Some other semantic analysis technique may be used to capture different fragments but that produces similar results.

4 EXPERIMENT

Our work is presented here as an experimental process. It follows the guidelines by Wohlin et al. (2000). In this section, we start introducing the experiment definition and planning. The following sections, will direct to the experiment execution and data analysis.

4.1 Goal Definition

Our goal is to compare clone findings between two private source code repositories, one with Object-Oriented code and other with Procedural Projects, using an exact-similarity threshold.

To achieve this, we are going to execute an experiment in a controlled environment using a clone detection tool. This comparison test attempts to answer questions about clone incidence related to programming language paradigms.

The goal is formalized using the GQM Goal template proposed by Basili and presented in (Solingen and Berghout, 1999):

- Analyze our corporate projects
- with the purpose of evaluation OO Systems against Procedurals Systems
- with respect to code clone manifestation
- from the point of view of the programmers
- in the context of an environment with a well-defined software process

4.2 Planning

Context selection: The experiment will be off-line and executed with the CloneDR clone detector inside a Java and a PL/SQL code repository containing about seven different systems each. The selected subject organization is an educational-purpose company active in market since the 60s, with more than 2,000 employees and around 50,000 customers. The PL/SQL development team differs from the java team by more experience and job constancy, as shown in Table 1. PL/SQL team consists of 4 developers with age from 37 to 40 years old and an experience average of 15 years against 5 developers for java team, starting with 23 years old, most with only a 2-year programming experience. About 20 systems are maintained by procedural language team and 11 systems by OO team. Deadline pressure levels for both teams are the same.

Table 1: Experience and Constancy of Development Teams.

TEAM	DEVELOPER ID	AGE	YEARS OF LANGUAGE EXPERIENCE	YEARS ON COMPANY
PL/SQL TEAM	1	40	18	20
	2	43	15	17
	3	38	13	15
	4	37	8	11
JAVA TEAM	5	43	13	16
	6	29	8	4
	7	27	7	2
	8	25	6	2
	9	23	5	2

Hypothesis formulation: The research question for this experiment is: Have object-oriented software systems more efficiency than procedural systems, regarding code clone manifestation?

Since private organizations provide a more controlled environment to adopt standardization of software development, we are interested about differences in the incidence of clones within programming language paradigm code repositories.

We will compare some extracted statistics of our Java systems with seven other PL/SQL private systems from our target corporation using the same extraction tool, respecting the similarity threshold between comparisons.

To assure the reliability of our hypothesis test, we will calculate the average between the proportional results of exact similarity for each system (S), where similarity threshold is 1 (means 100% or exact clones). The proportion (P) is calculated by dividing Cloned Source Lines of Code or Clone Sets (C) by its respective total of Source Lines of Code (SLOC).

When defining the variables for the formal test, the systems size was considered, because just the clone numbers does not imply conditions to evaluate a greater propensity to lower abstraction. Besides, the similarity threshold as 1 indicates an identical clone, evidencing more reliably the possibility of a type of Technical Debt (DT) (Guo et al., 2011)such as failure to code reuse or failure to use Abstract Data Types (ADT).

Capture of Clone Sets were included in our experiment in order to identify repositories storing methods that are cloned in excess.

Keeping this idea, we will try to reinforce the following hypothesis:

HYPOTHESIS 1

- Null hypothesis H0SL: Object-Oriented Systems (1) have same incidence of Cloned SLOC than Procedural Systems (2) in the context of our corporate projects.
  - H_0^{SL}: $\mu 1_{(\text{Cloned SLOC Proportion})} = \mu 2_{(\text{Cloned SLOC Proportion})}$
- Alternative hypothesis H1SL: Object-Oriented Systems (1) have lower incidence of Cloned SLOC than the Procedural Systems (2) in the context of our corporate projects.
  - H_1^{SL}: $\mu 1_{(\text{Cloned SLOC Proportion})} < \mu 2_{(\text{Cloned SLOC Proportion})}$

HYPOTHESIS 2

- Null hypothesis H0CS: Object-Oriented Systems (1) have same incidence of Clone Sets than Procedural Systems (2) in the context of our corporate projects.
  - H_0^{CS}: $\mu 1_{(\text{Clone Sets Proportion})} = \mu 2_{(\text{Clone Sets Proportion})}$
- Alternative hypothesis H1CS: Object-Oriented Systems (1) have lower incidence of Clone Sets than the Procedural Systems (2) in the context of our corporate projects.
  - H_1^{CS}: $\mu 1_{(\text{Clone Sets Proportion})} < \mu 2_{(\text{Clone Sets Proportion})}$

Independent variables: AST method; Our Object-Oriented and Procedural Industrial Projects, written respectively in Java and PL/SQL. Moreover, the parameters used to configure the tool used on this experiment will be described in Section 5.

Dependent variables: The Clone Sets and Cloned SLOC proportions (PS) and averages (μ) between results of Cloned SLOC and Clone Sets (CS) and their respective SLOC will be used as dependent variables. They are described as follows:

- **Proportion**: $P_S = C_S/SLOC$
- **Final Average**: $\mu = (P_{S1} + P_{S2} + \ldots + P_{Sn}) / n$

Objects selection: The selection of Object-oriented and procedural projects is shown in Table 2, describing their names, amount of LOC and the kind of repository they belong to. The private code projects size varied from a 4.7K SLOC to a 102K SLOC application. This selection was done by convenience. We have used some corporate projects which we were clone consultants for. The analysis is non-intrusive to developers as the data were drawn directly from the code repository, they did not know which source code would be extracted.

Instrumentation: We have used CloneDR tool described in section 3. Results are printed to the standard output. Additional information results are exported to HTML files in the same directory of the original system source.

Table 2: Overview of selected projects.

REPOSITORY	PROJECT NAME	SLOC
OO	Graduação (Academic System)	26462
	Concurso (Contest System)	6719
	Extensão (Extension System)	6693
	Pagamento (Payment System)	7743
	Portal (Web Portal System)	9648
	Pós-Graduação (Post-grad System)	4755
	Protocolo (Protocol System)	13739
PROCEDURAL	Concurso (Contest System)	14059
	EAD (Distance Learning System)	56406
	Professor System	18744
	Protocolo (Protocol System)	53298
	Graduação (Academic System)	102223
	Financeiro (Finance System)	68553
	Pós-Graduação (Post-grad System)	29735

5 EXPERIMENT OPERATION

In this section, we describe the whole experiment execution. The detection tool was configured to consider only functions or methods with a minimum of 6 LOC. We do not analyze in this work clone distribution and localization over files or directories.

5.1 Execution

First, we extracted clone information for the whole OO repository to compare with the Procedural Repository results, using the CloneDR tool. Then, each project was analyzed individually still with the same tool and every clone-related discovered information was recorded and analyzed by hand.

At first we can confirm that inside our PL/SQL repository there are fewer clone manifestations than the Java repository. We may see a mean of proportional Cloned SLOC for the Procedural repository of 11,20%, meanwhile inside the OO repository the mean is 19,54% using the highest similarity threshold value.

5.2 Data Validation

The CloneDR clone detection tool generated HTML reports where we extracted the cloned methods to validate by hand a sample of the cloned methods. This brings more confidence on what was analyzed by the clone detection tool.

To ensure analysis, interpretation and validation, we used two types of statistical tests: Shapiro-Wilk Test and the T-Test. Shapiro-Wilk test, normally applied on smaller populations, was used to verify normality of the samples. The T-Test was used to check our hypothesis. All statistical tests were performed using the SPSS tool (SPSS, IBM).

Table 3: CloneDR statistics for OO and Procedural code repositories.

PARADIGM	PROJECT NAME (*S*)	SLOC	CLONED SLOC		CLONE SETS	
			C_S	P_S	C_S	P_S
Object-Oriented	Extensão (Extension System)	6693	691	10.32	29	0.43
	Concurso (Contest System)	6719	902	13.42	31	0.46
	Pós-Graduação (Post-grad System)	4755	678	14.26	31	0.65
	Pagamento (Payment System)	7743	1706	22.03	51	0.66
	Portal (Web Portal System)	9648	2202	22.82	75	0.78
	Graduação (Academic System)	26462	6359	24.03	209	0.79
	Protocolo (Protocol System)	13739	4106	29.89	112	0.82
Procedural	Concurso (Contest System)	14059	966	6.87	76	0.54
	Pós-Graduação (Post-grad System)	29735	2506	8.43	169	0.57
	EAD (Distance Learning System)	56406	5002	8.87	335	0.59
	Professor System	18744	2150	11.47	142	0.76
	Financeiro (Finance System)	68553	9459	13.80	453	0.66
	Graduação (Academic System)	102223	14692	14.37	822	0.80
	Protocolo (Protocol System)	53298	7786	14.61	420	0.79

6 RESULTS

To answer our experiment question, we executed all individual tests and created a table showing data to compare with the results obtained from the experiment. The Table 2 already showed several statistics collected from the analysis of our experiment.

6.1 Analysis and Interpretation

Clone detection statistics from all the OO Projects analyzed are also present on Table 3.

The values on "Cloned SLOC" and "Clone Sets" are representing the results after an analysis using an exact-similarity threshold. The Procedural projects presented significantly much more SLOC than OO systems. The "P_S" column represents the proportional values for the clone detection, for the respective system.

Analyzing the Table 3, we note that clone incidence is not related to the project size. The bigger the worst does not apply here, since we have the Java version of Academic System with 24% Cloned SLOC inside 26K SLOC versus the PL/SQL version presenting 14% Cloned SLOC for 102K SLOC.

PL/SQL Contest System showed excellent results in comparison to the OO Protocol System, both having about 14K SLOC. The OO Protocol System returned the higher Clone Set value, which indicates a worse use of methods abstraction. This system also had the worst performance, with almost 30% of cloned code.

Academic and Protocol System were the top-cloned software. Besides having a huge number of SLOC, they are maintained by a vast and heterogeneous development team.

For the OO, Extension System was the more clone-free project. The Procedural system with less proportionally Cloned SLOC was the Contest System, with among 7% of exact clones. The final average found for the OO and Procedural Cloned SLOC and Clone Sets can be found on Table 4.

Table 4: Final average results.

FINDINGS	PARADIGM	
	OO	PROCEDURAL
CLONED SLOC	19.54	11.20
CLONE SETS	0.66	0.67

Based on these results, we observe that was some significant difference between the two kinds of repository. The Object-oriented Systems showed more proportional clone incidence than the Procedural ones. With this data, is not possible yet to make any assumption about results without sufficiently conclusive statistical evidence.

Firstly, we applied the Shapiro-Wilk test with a significance level of 0.05, analyzing the distribution normalization. The Sig variables (also known as p-values) for Cloned SLOC were 0.615 on OO samples and 0.261 on Procedural samples. For Clone Sets, the p-values were 0.216 on Procedural Systems and 0.193 on OO. The numbers on all samples for each hypothesis were above the significance level, so, we assume that data distribution is normal.

Applying the T-Test (Figure 2), we obtained a Sig. result of 0.014 for Cloned SLOC samples and 0.818 for Clone Sets. Only the p-value for Clone Sets was above the significance level of 0.05. This means that, regarding Cloned SLOC, we cannot assert the null hypothesis for $H0^{SL}$. In other words, the differences of cloned single lines of code found on object-oriented programs was relatively higher than the numbers returned from procedural systems.

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
SLOCS	Equal variances assumed	7,002	,021	2,867	12	,014	8,33571	2,90706	2,00177	14,66966
	Equal variances not assumed			2,867	8,360	,020	8,33571	2,90706	1,68190	14,98953
CSETS	Equal variances assumed	,566	,466	-,236	12	,818	-,01714	,07279	-,17573	,14145
	Equal variances not assumed			-,236	10,716	,818	-,01714	,07279	-,17787	,14358

Figure 2: T-Test results. Exported from IBM SPSS.

The Levene's Test is used to test if the samples have equal variances, also called homogeneity of variance. The sig value for CSETS is 0.466 (higher than 0.05) which means that, for Clone Sets the scores do not vary too much. Observing Source Lines of Code, the sig value is 0.021 (less than 0.05). Because of this, for SLOCS there is a statistically significant difference between the means.

For the Clone Sets null hypothesis $H0^{CS}$, the final decision is to not reject it. In fact, for Clone Sets there was a strong retention for the null hypothesis $H0^{CS}$ ($\mu1$(Clone Sets Proportion) = $\mu2$(Clone Sets Proportion)). In real terms, there is a probability of almost 82% that we will mistakenly reject the similar Clone Sets incidence, although Object-Oriented coding has features that make easy code abstraction and reuse.

The results indicate that the use of abstraction for both Procedural and OO programs in this organization present a similar efficiency. We have more Cloned SLOC for OO than Procedural projects, but when implementing abstraction in functions or methods, the clone findings are almost equal. This means that, although object-oriented languages provide means to a better use of abstraction (e.g. polymorphism), the analyzed Java repository showed an inadequate behavior for this issue.

The development teams are different for PL/SQL and Java. The PL/SQL team has a characteristic of having lower staff turnover than Java team. Thus, the procedural repository takes advantage of owning maintainers with more experience time inside the company, with a solid knowledge about the business rules and knowing more deeply the code.

Moreover, is evident that, even with design patterns and frameworks adopted by the OO team, experience may have great influence on the capacity of abstraction. In background, there is a warning for the software management acting with regard to recycling and adoption of good practices by the teams.

For the organization, these results require further study about other causes that may have compromised the quality of coding. Features concerning different development patterns or different team profiles as age, maturity and knowledge could be studied to check their interference on clones' manifestation.

6.2 Threats to Validity

In spite of the fact that our corporate systems are a mature, real world, large projects, and our results seem to be quite consistent with the systems sizes, our study shows threats to its validity that we must consider:

- We cannot conclude that all closed-source projects will present similar results as ours. Process maturity can play a large role on code clone manifestation;
- Other software characteristics such as complexity may affect the results. We have not test for those variables;
- Adoption of design patterns also may influence on code clone manifestation;

The profile of the development team (team size, age, experience) also can represent a change on the final sample.

7 CONCLUSIONS AND FUTURE WORK

In our experiment, we analyzed two different repositories, which comprise systems of distinct programming language paradigms and found evidences that clone incidence is not directly related to the size of code. In fact, the studied Procedural systems had fewer lines of cloned code with much more coding lines than the OO ones.

The lack of code abstraction ended up being similar in both cases. Questions about the profile of both Java and PL/SQL development teams must be asked to check if experience, age, instruction degree and other factors, may affect the coding maintainability.

We encourage more research inside private environments to test hypothesis only studied on Open Source Software systems. Also, our corporate Object-Oriented Systems had very few SLOC than other Object-Oriented Open Source Systems. It is important to replicate this experiment inside several other private repositories to check if they present the same behavior. The more the systems are tested, more we assure external validity.

As mentioned before, we adapted the software engineering experimental process described in Wohlin et al (2000) to clones mining experiments. We believe that the studies, applications, and tools for software clone mining can benefit from this type of approach. Rigorous experimental description facilitates replication of studies and the executing of systematic reviews and other types of secondary analysis.

As future work, we have in mind a few projects related to clone incidence. The first one is to verify if the human profile of development team has some direct effect on clone appearance. Data like age, experience and qualification may be extracted and combined from several sources to mount this profile. Other insight is to explore code comments to find out words that indicate something that was purposely implemented missing some pieces (for many reasons) and this will have to be done some time, indicating a Technical Debt (TD) issue.

ACKNOWLEDGEMENTS

This study could only be developed due to the support of Tiradentes University – UNIT, along with the Technology Information Department – DTI, who provided the repository used in our experiment.

REFERENCES

I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in ICSM, 1998,pp. 368–377.

Y. Ma and D. Woo. Applying a Code Clone Detection Method to Domain Analysis of Device Drivers. In Proceedings of the 14th Asia Pacific Software Engineering Conference (APSEC'07), pp. 254–261, Nagoya, Japan, December 2006.

A. Marcus and J. I. Maletic. Identification of high-level concept clones in source code.In Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01), pp. 107-114, San Diego, CA, USA, November 2001.

S. Khatoon, A. Mahmood, and G. Li, "An evaluation of source code mining techniques," Proc. - 2011 8th Int. Conf. Fuzzy Syst. Knowl. Discov. FSKD 2011, vol. 3, pp. 1929–1933, 2011.

C. K. Roy and J. R. Cordy, "An Empirical Study of Function Clones in Open Source Software," 2008 15th Work. Conf. Reverse Eng., pp. 81–90,2008.

J. R. Cordy and C. K. Roy, "The NiCad Clone Detector," 2011 IEEE 19th Int. Conf. Progr. Compr., no. Figure 3, pp. 219–220, 2011.

N. Schwarz, M. Lungu, and R. Robbes, "On how often code is cloned across repositories," Proc. - Int. Conf. Softw. Eng., pp. 1289–1292, 2012.

D. Rattan, R. Bhatia, and M. Singh, Software clone detection: A systematic review, vol. 55, no. 7. Elsevier B.V., 2013.

M. Kim, V. Sazawal, and D. Notkin, "An empirical study of code clone genealogies," ACM SIGSOFT Softw. Eng. Notes, vol. 30, p. 187, 2005.

C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Sci. Comput. Program., vol. 74, pp. 470–495, 2009.

D. Rattan, R. Bhatia, and M. Singh, Software clone detection: A systematic review, vol. 55, no. 7. Elsevier B.V., 2013.

C. J. Kapser and M. W. Godfrey, "'cloning considered harmful' considered harmful: Patterns of cloning in software," Empir. Softw. Eng., vol. 13, pp. 645–692, 2008.

Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslén (2000). Experimentation in software engineering: an introduction. Kluwer Academic Publishers, ISBN: 0-7923-8682-5.

M. Colaço, M. Mendonça, M. André, D. F. Farias, and P. Henrique, "A Neurolinguistic-based Methodology for Identifying OSS Developers Context-Specific Preferred Representational Systems," Context, no. c, pp. 112–121, 2012.

OpenSource.com What is open source? Retrieved from https://opensource.com/resources/what-open-source

Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. Da Silva, A. L. M. Santos, and C. Siebra, "Tracking technical debt - An exploratory case study," IEEE Int. Conf. Softw. Maintenance, ICSM, pp. 528–531, 2011.

R. van Solingen and E. Berghout (1999). The Goal/Question/Metric Method: A practical guide for quality improvement of software development. McGraw-Hill.

SPSS, IBM Software, http://goo.gl/eXfcT3

C. K. Roy and J. R. Cordy, "A Survey on Software Clone Detection Research," Queen's Sch. Comput. TR, vol. 115, p. 115, 2007.