2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)

# Collection and Analysis of Source Code Metrics for Composition of Programming Learning Profiles

Francisco Alan de Oliveira Santos
Post-Graduate Program in Computer Engineering and Systems
State University of Maranhão
São Luis, Brazil
alansantospi@gmail.com

Luis Carlos Costa Fonseca
Post-Graduate Program in Computer Engineering and Systems
State University of Maranhão
São Luis, Brazil
luiscarlos@uema.br

*Abstract*— **This paper presents an approach for the application of clustering algorithms to uncover computer programming learning profiles by using evidence extracted from source code metrics. A system for automatic assessment of programming activities featuring capture of source code metrics was developed, in order to build a dataset containing metrics extracted from programs developed by beginners in a computing course. The dataset was submitted to three clustering algorithms. The results were promising when clustering students according to these indicators using the K-means algorithm.**

***Keywords-component; software metrics; source code analisys; programming learning; data mining; clustering***

## I. INTRODUCTION

The learning of computer programming depends on the absorption of various related knowledge [10] and involves the combination of several cognitive abilities, so that the evaluation of this knowledge and skills requires careful analysis [12]. A common way to evaluate that knowledge is by reviewing the code produced by students. In this context, alternatives for the automation and execution of source code tests have arisen [3]. In general, such proposals include several tools for the assessment of programming activities, whose adoption has commonly been used integrated to Virtual Learning Environments (LVE).

The advent of these tools represents an opportunity of research in Data Mining, since it is possible to record the history of all the programs developed by the students. Therefore, a large volume of relevant data can be generated, since the codes submitted by students are artifacts from which information about their learning evolution can be extracted.

This work presents a process of data collection and analysis regarding achievement in programming for statistical analysis of student performance. To fulfill this purpose, source code metrics capture functionalities have been integrated into an automatic programming exercise evaluation environment. The data collected were submitted to clustering algorithms in order to identify student profiles. These procedures were performed in order to answer the following question: can the data generated from assisted analysis of source code be used to generate objective evidence about programming learning? From this questioning, the hypothesis of this research was elaborated, namely: The metrics derived from the automatic analysis of source code have the potential to generate a set of numerical values, whose mining can help finding patterns to compose programming learning profiles.

## II. THEORETICAL REFERENCE

Systems for automating and executing source code tests have evolved considerably [9]. However, according to [7], there are still considerable shortcomings with regard to the effectiveness of these tools, especially in terms of ascertaining the achievement of the desired educational goals. For [4], these systems need to evolve to provide greater direction from a pedagogical perspective.

The application of techniques of classification, measurement and qualification of large quantities of educational data has been called Learning Analytics (LA) [11]. According to [11], LA provides means to obtain information about student interaction with learning resources. One of its focuses is the analysis of automatically captured data to improve learning by monitoring academic performance. In turn, [8] consider the use of software metrics based on student code as a revolutionary strategy for teaching.

## III. RELATED WORK

[2] presented BitFit, an activity submission tool from which a predictive model based on binary classification was constructed to identify the probability of a student being disapproved.

[6] proposed a set of software metrics based on the static analysis of student codes. The study found that metrics correlate with students' grades.

[5] proposes the use of metrics to determine which students may be at risk at the beginning of a course from the logs of programming activities generated by a programming environment. Logs were used to generate a set of variable data, such as time and quantity of errors.

## IV. METHODOLOGY

A study on the main requirements for the use of metadata for automatic correction of programming tasks was performed. Then, the metrics capture functionalities were implemented. Subsequently, a case study was conducted in order to ascertain the level of effectiveness of this approach.

2161-377X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICALT.2019.00056

With this set, attributes selection techniques were adopted to identify the most important evidences for the data clustering.

## V. Case Study

A source code repository with practical programming assessments has been analyzed. The tool chosen for collecting the data is CodeTeacher [10], a system for automatic assessment of programming exercises in Java language. CodeTeacher has been evolved to version 2.0, which adds mechanisms for calculating various source code metrics present in a given repository of student software solutions for programming exercises submitted to the tool.

An assignment in Java programming language has been proposed to classes between 2016.2 and 2018.1 in order to obtain a subset of metrics, totaling 72 instances. The assignment involved writing a main method to serve as a program entry point for using logical and relational operators, implementing Java methods, understanding the use of loop structures, and invoking methods across objects.

After the preprocessing stage, we performed the attribute selection process, in which search algorithms were applied in the Weka tool in order to find the most relevant subsets of attributes. Table I shows a list of metrics selected, since they have been shown to be more significant.

TABLE I. Source Code Metrics

| Metric | Description |
|---|---|
| Non Commenting Source Statements (NCSS) [1] | Determines complexity of files by counting the source lines which are not comments. |
| File Length | Average source file length. |
| Package Declaration | Number of classes that have a package declaration, and (optionally) whose package name matches the directory name for the source file. |
| NPath Complexity | Computes the number of possible execution paths through a function. |
| Local Variable Name | Checks that local, non-final variable names conform to a specified format. |
| Method Count | Number of methods declared in each type declaration by access modifier or total count. |
| Cyclomatic Complexity Number (CCN) | Also known as McCabe Metric [1]. The complexity is measured by the number of 'if', 'while', 'do', 'for', '?:', 'catch', 'switch', 'case' statements, and operators '&&' and '\|\|' in the body of a constructor, method, static initializer, or instance initializer. |
| Outer Type Number | Number of types declared at the outer (or root) level in a file. |
| Magic Number | Counts magic number ocurrences. A magic number is a numeric literal that is not defined as a constant. |
| Whitespace Around | Number of tokens surrounded by whitespace. |
| Indentation | Counts incorrect indentation of Java code. |
| Package Name | Number of packages whose names conform to a specified format. |
| Whitespace After | Checks the padding of an empty for initializer; that is whether white space is required at an empty for initializer, or such white space is forbidden. |
| Method Length | Average length of methods and construtors. |
| Executable Statement Count | Number of executable statements. |

In order to guarantee a pedagogical perspective, the metrics were mapped to the evaluation criteria, these, in turn, are related to the learning objectives contained in the curricular matrix of the course, in that sense, the metrics were distributed as follows: Code structure, Language conventions, Naming patterns and Difficulty and complexity.

## VI. Cluster Analisys and Validation of the Best Algorithm

Cluster analysis is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). There are several approaches to clustering, however, we considered only hierarchical agglomeration and partitioning.

The choice of the appropriate clustering algorithm for the created dataset was made by internal cluster validation, which measures the connectivity, silhouette width, and Dunn index. These internal measures were simultaneously calculated for multiple clustering algorithms in combination with a series of cluster numbers. The algorithms "hierarchical", "k-means" and "pam" were tested, and k-means proved to be the best option. The determination of the number of clusters, in turn, was validated by the analytical method of minimizing the sum of the squares of the clusters, which remained stable from eight segments. However, this number of clusters would make it difficult to interpret the results, so a dendogram was created as an auxiliary visual mechanism. With this procedure, the four-segment option proved to be a good choice for the number of clusters in this analysis model.

## VII. Threats to Validity

It is important to highlight the limitations of this work that represent potential threats to the validity of the research, among which, we can mention:

- Small size and dimension of the dataset
- Number and profile of chosen indicators not appropriate enough to determine effective student segmentation;
- Small number of clusters;

## VIII. Discussion of Results

Considering the characteristics of the solutions, evidence was identified that showed degrees of difficulties on a larger and smaller scale. Table II shows four distinct groups, according to the mean values shown in Table III.

TABLE II. PROFILE CLASSES

| Cluster | Code Structure | Language Conventions | Naming Patterns | Difficulty and complexity |
|---|---|---|---|---|
| A | Above | Below | Above | Above |
| B | Below | Far above | Below | Above |

| | | | | |
|---|---|---|---|---|
| C | Below | Below | Below | Above |
| D | Below | Below | Below | Far below |

TABLE III. CLUSTER DATA MEANS

| Cluster | A | B | C | D |
|---|---|---|---|---|
| **Size** | 14 | 21 | 10 | 26 |
| **NCSS** | 11.24 | 9.45 | 13.50 | 10.12 |
| **File Length** | 24.93 | 20.27 | 26.52 | 18.54 |
| **Package Declaration** | 3.21 | 0.05 | 1.90 | 2.77 |
| **NPath Complexity** | 0.63 | 0.94 | 2.07 | 1.19 |
| **Local Variable Name** | 0.00 | 0.00 | 1.00 | 0.00 |
| **Method Count** | 3.21 | 3.24 | 2.10 | 3.04 |
| **Cyclomatic Complexity** | 1.31 | 1.47 | 2.03 | 1.59 |
| **Outer Type Number** | 3.21 | 3.29 | 2.10 | 3.00 |
| **Magic Number** | 3.50 | 3.81 | 2.90 | 3.96 |
| **Whitespace Around** | 9.14 | 8.90 | 15.90 | 12.38 |
| **Indentation** | 28.29 | 6.05 | 24.80 | 8.12 |
| **Package Name** | 0.00 | 1.71 | 0.20 | 0.00 |
| **Whitespace After** | 0.86 | 0.76 | 0.90 | 0.69 |
| **Method Length** | 20.27 | 12.71 | 18.82 | 14.47 |
| **Executable Statement Count** | 30.43 | 20.95 | 20.90 | 23.35 |

The Difficulty and complexity category was recognized mainly by the presence of low values in the columns of effort and difficulty metrics. When checking the solutions, Group A, for example, identifies excess lines of code, instructions, conditional structures, and a large number of comparisons. This indicates problems with the correct use of operators and logical expressions. conditional and repetitive control structures. These issues address learning difficulties due to excessive code, as they demonstrate the low ability of students in this group to solve simple programming problems succinctly.

The language conventions and naming standards category provide indicators of difficulties in good programming practices. Students with high values in this category present difficulties in learning about language conventions by the very high rates in the values of several metrics related to this item.

Regarding the structuring of the code, high values were found mainly referring to the indentation of the code, that is to say that several occurrences of indentation problems were detected, besides cases of whitespace in unnecessary places, which justifies the low income of students in this respect.

## IX. CONCLUDING REMARKS

A preliminary version of an automatic code analysis tool was developed, including metrics collection functionality as an auxiliary tool in the teaching-learning programming process. The data collected by CodeTeacher were submitted to the Simple K-Means algorithm for grouping of learning profiles.

According to the analyzes performed, clustering algorithms and information visualization tools, the presented results demonstrate the viability and importance of the learning analysis by software metrics. With this approach, it is possible to lead programming teachers to understand learning difficulties of their students.

As future work, we intend to integrate mechanisms of data clustering and information visualization to CodeTeacher. We believe that the insertion of these functionalities collaborates to build a more complete tool to aid the practical evaluation of programming, and thus can extend the teaching-learning experience of programming, functioning as an enriching element of this process.

## REFERENCES


[1] Curtis, Bill et al. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Transactions on software engineering, n. 2, p. 96-104, 1979.

[2] Estey, A. and Coady, Y. (2016). Can interaction patterns with supplemental study tools predict outcomes in cs1? *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '16*, pages 236–241.

[3] Ebrahimi, A. (1994). Novice programmer errors: language constructs and plan composition. International Journal of Human-Computer Studies, 41(4):457 – 480.

[4] Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli Calling '10, pages 86–93, New York, NY, USA. ACM.

[5] Munson, Jonathan P. (2017) Metrics for timely assessment of novice programmers. Journal of Computing Sciences in Colleges, v. 32, n. 3, p. 136-148.

[6] Otero, J., Junco, L., Suarez, R., Palacios, A., Couso, I., and Sanchez, L. (2016). Finding informative code metrics under uncertainty for predicting the pass rate of online courses. 373:42–56.

[7] Oliveira, M.; Nogueira, M.; Oliveira, Elias (2015). Sistema de apoio à prática assistida de programação por execução em massa e análise de programas. In: XIV Workshop de Educação em Computação (WEI)-SBC 2015.

[8] Peterson, A., Spacco, J., and Vihavainen, A. (2015). An exploration of error quotient in multiple contexts. Proceedings of the 15th Koli Calling Conference on Computing Education Research, pages 77–86.

[9] Romli, R.; Sulaiman, S.; Zamli, K. (2010) Automatic programming assessment and test data generation a review on its approaches. In Information Technology (ITSim), International Symposium in, v. 3, p. 1186 –1192.

[10] Santos, Francisco Alan; Segundo, Plácido; Telvina, Mardoqueu (2017). CodeTeacher: Uma Ferramenta para Correção Automática de Trabalhos Acadêmicos de Programação em Java. In: VI Congresso Brasileiro de Informática na Educação, Recife. Porto Alegre: Sociedade Brasileira de Computação SBC, 2017. v. 1. p. 1152-1161.

[11] Sclater, N. (2017). Learning Analytics Explained. 1. ed. New York: Routledge.

[12] Tobar, C. M. et al (2001). Uma Arquitetura de Ambiente Colaborativo para o Aprendizado de Programação. XII Simpósio Brasileiro de Informática na Educação, Vitória, ES.