

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900

Open access books available

185,000

International authors and editors

200M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Automatic Mapping of Student 3D Profiles in Software Metrics for Temporal Analysis of Programming Learning and Scoring Rubrics

*Márcia Gonçalves de Oliveira, Ádler Oliveira Silva Neves
and Mônica Ferreira Silva Lopes*

Abstract

The purpose of this chapter is to present an online system for a 3D representation of programming students' profiles on software metrics that quantify effort and quality of programming from the analysis of source codes. In this representation, each student profile is a three-dimensional vector represented by a set of programming solutions developed by a student and mapped on 348 metrics of software during a programming course. Applying this profile representation, we developed a system with the following functionalities: generation of student's timelines to verify the evolution of metrics in a sequence of programming solutions over a course, different visualizations of these variables, automatic selection of representative codes for composition of rubrics with less effort of evaluation and selection of metrics that more influence in scores attributed by teachers. The advantages of this system are to enable the analysis of where the learning difficulties begin, the monitoring of how a class evolves along a course and the dynamic composition of rubric representations to inform assessment criteria. The system proposed therefore presents itself as a relevant tool to assist teachers about decisions of an evaluative process, allowing in fact to assist students from the beginning to the end of a course.

Keywords: learning analysis, programming, software metrics, learning profiles

1. Introduction

Analysis of programming learning for purposes of assisting and qualifying a learning process from beginning to end represents an onerous task to programming practice, since the practice of assisted programming spends time and effort in activities, assessment, especially when there is the application of a lot of exercises and there are many students in a class. Thus, applying learning analysis that makes it possible to compare programming solutions developed by different students and to verify how a student's solution evolve over time represent a real challenge for the evaluation of programming.

Although there are already several solutions for representing and comparing programming students' profiles [1, 2], there are few solutions for a temporal analysis of the learning of these students during a programming course.

A more recent proposal to analyze programming learning aims to map source codes into software metrics that quantify effort and quality of programming [3]. Through these metrics, for each programming activity, it is possible to compare student's solutions under different variables to identify classes of solutions, common learning difficulties, good practices of programming and even plagiarism.

Although the proposal of [3] makes it possible to compare student profiles of a class in each programming activity, it is laborious for a teacher through this instrument to verify how these evaluation metrics evolve over time, that is, to each activity of a course, for each student. This type of monitoring allows the programming teacher to identify in which students develop better in their learning processes and where students begin to present learning difficulties.

In order to meet this need by offering programming teachers an instrument to monitor the learning process of their students, this chapter extends the proposal of [3] generating 3D views of student profiles mapped into selected software metrics. These metrics characterize each student's efficiency, style, and programming effort with each programming solution they develop over a course.

In addition to the 3D representation to analyze learning, this system selects dynamically programming solution samples for a teacher to score until finding a representative set of rubric representations to inform evaluation criteria. This functionality may contribute later to generate a representative set of programs to train automatic assessment system of programming exercises.

Another feature of this system that is still in the testing phase is the prediction of students' performances in an activity based on their history of solving activities or the solutions of that same activity developed by other students.

The main contribution of this chapter is, therefore, offering a tool to support evaluation, decision-making in the field of programming, enabling teachers to analyze and monitor their students' learning for each programming activity under a wide range of variables, anticipating a predictable future of poor performance.

In order to present the fundamentals and the functionalities of the proposed system, this chapter is organized in the following order. Section 2 presents the related work. Section 3 describes the system architecture with 3D representations of profiles and the selection of rubric representations. Section 4 highlights reports of application of our system in a programming distance course. Section 5 concludes this work highlighting the main results, future work and final considerations.

2. Static analysis of programming

Static analysis is an automatic assessment approach to programming learning based on analysis of code. Through static analysis, it is possible to analyze effort, complexity, efficiency and quality of programming [4–6].

The main advantages of static analysis are lower cost, less reliance on the teacher's reference solution and the possibility of offering an evaluation closer to human evolution, although many programming teachers have prioritize the dynamic analysis, which is an analysis based on the correct and efficient execution of programs. Static analysis can therefore be used in the analysis of programming codes for the following purposes:

- Effort measurement and coding complexity [4]
- Prediction of performance [7, 8]
- Programming style analysis [5, 9]
- Evaluation by software metrics [3, 10]
- Recognition of signs of plagiarism [3]
- Recognition of rubrics [11]
- Recommendation of activities [1]
- Programming proficiency analysis [12]
- Analysis of learning difficulties and good programming practices [3].

Among the technological solutions of programming learning analysis based on static analysis already proposed, we highlight: the metrics of Halstead and McCabe [4, 5, 13], the evaluation of programming skills by software metrics [10], the recommendation system of activities according to learning difficulties [1], the analysis of difficulties by software metrics [3], the evaluation of how programming students learn from the analysis of their programming codes [14] and the programming proficiency analysis of SCALE system [12].

2.1 The evolution of static analysis strategies of programming

The main static analysis strategies of programming developed from the 1960s to the present day were based on software evaluation metrics that evolved from the purposes of measuring codes and software quality for educational purposes of diagnosing learning difficulties and evaluating difficulties, skills and even programming skills.

In the 1970s and 1980s, the software metrics were used to analyze programming codes for the purposes of estimating effort, complexity and programming style. Thus, some developed strategies were associated the programming process with the psychological complexity to evaluate performance in programming without necessarily having the concern to help those who had more difficulties [4, 9].

During the 90s until the year 2010, strategies of static analysis based on metrics for learning analysis, but in times when the Intelligent Tutoring Systems (ITS) were high, it was sought to represent the model or profile of a student, focusing more on his learning.

In more recent research on programming learning analysis, in addition to having a concern to better understand the students' learning profiles, there have been attempts to remedy learning difficulties [3]. Other trends in programming learning analysis are proficiency assessment [12], prediction of performances [15] and the classification of profiles by learning levels [2].

2.2 Related works

The main related works to our proposal are the assessment system based on the software metrics of [3], the instruments of visualization of programming students' profiles of [16], the recognition strategy of profiles by source code analysis metrics of [2], the selection model of features of [17], the system of recognition of rubrics

with dimensionality reduction of [11] and the study of [18] involving the discovery of longitudinal patterns.

PCodigo II is an online system of automatic mapping of students' profiles in software metrics to analyze programming learning [3]. In addition to profiling mapping in 348 software metrics, PCodigo II has massive execution, similar profile graphing, information visualization, and plagiarism analysis capabilities.

The first applications of PCodigo II of [3] in real programming exercises demonstrate the effectiveness of this system for the diagnostic assessment of programming learning. Thus applying PCodigo II in real programming exercises it was shown that teachers, taking into account what the metrics say, can recognize the learning difficulties, good programming practices and classes of learning profiles of a whole class in a fast, detailed and holistic way.

The chapter of [16] presents some information visualization instruments in a multidimensional perspective to help teachers in the analysis of programming learning with mapping of profiles on software metrics. Through generated visualizations, we can analyze and compare profiles under different variables to recognize learning difficulties and classes of solutions from similar characteristics.

The strategy of profile recognition by static analysis of codes based on metrics of [2] aims to infer profiles of programmers from analysis of their Java code, classify them according to skills and continually evaluate their progress in the practice of programming in a course. The detected profiles are a novice, advanced beginner, proficient and expert.

Some metrics used in this strategy are a number of sentences, conditional control and repetition structures, types of data, classes, operators, lines of code, and other code. The advantage of this strategy in relation to our system is to classify and qualify students. However, we automatically select the most appropriate metrics to evaluate each type of programming solution.

For an automatic selection of evaluation variables, we highlight the selection model of the characteristics of [17], which combines clustering techniques and algorithm to create a feature map by selecting relevant terms in the texts of the groups of notes of the evaluation of a teacher. In our proposal, the relevant characteristics, that is, the most important metrics for each programming solution, we can visualize through heat maps comparing different solutions from five or more software metrics.

Regarding the composition of rubrics, a strategy to highlight is the proposal of [11], which is based on clustering and Principal Component Analysis techniques to recognize, from solutions developed by students, examples of solutions that represent, in a rubric scheme, the scores attributed by a teacher. This work complements these proposals by generating a ranking of samples of programming solutions for a teacher to score until finding the best set of rubric representations with a diversity of marks awarded.

According to [18], to understand how learning unfolds in the over time, it is necessary to move to a new learning perspective in which the units of analysis are separate but interrelated learning events.

Following this idea, the study of [18] investigates and validates longitudinal patterns in online participation as a measure to differentiate student performances.

The proposal of the system of this work, based on the study of [18], seeks to understand how programming learning unfolds and analyze longitudinal patterns.

In this way, following this proposal, in relation to the other Works Presented, we advanced in the 3D representation of profiles of programming students, in the view of characteristics represented by software metrics over time and the composition of rubrics from a ranking of selected solutions automatically for a teacher to score.

3. 3D representation system of programming students' profiles

The system of representation of profiles presented in this chapter is an evolution of *PCodigo II*, a software developed by which, by software metrics that quantifies effort and quality of programming, recognizes possible learning difficulties, good programming practices and until strong evidence of plagiarism among programs [3].

Thus our system extends the students' profiles representation of *PCodigo II* in a temporal dimension, selects more relevant metrics and allows the automatic selection of representative examples from a set of source codes for composition of rubric representation.

Figure 1 shows the system's architecture proposed in a scheme of inputs, processing and outputs come an integration of our system to the 1.0 and 3. x versions of Moodle virtual learning environment.

According to **Figure 1**, for version 1.9 of Moodle, the system receives as input a backup of Moodle's *Compacted Classroom* (in .zip, .rar, .gz or .tgz formats). For version 3. x of Moodle, the system is accessed through *Teacher's Credentials* to access a distance programming course of Moodle.

The course data imported from Moodle are as follows: student listing, activity listing, activity notes and *Submissions*, that are files of programming exercises. These data are then extracted by the *Extracting and Preprocessing* module and *Submissions* containing source codes that were written in C, C++, Java or Python languages are mapped to vectors whose dimensions are software metrics that quantify effort and quality of programming [3]. The submitted C programs are mapped on 348 software metrics and the Python programs, in 42 metrics.

Each vector representation on software metrics of a student's programming solution we call *Learning State*. Then, after generating *Learning States* of a programming class, the system gathers these representations in a *Cognitive Matrix* for analysis and comparison of programs written by students [3].

In order to analyze solutions in a generic way, we have reduced each *Learning State* to five metrics: *Maintainability*, *Cyclomatic Complexity*, *Indentation*, *Laconism* and *Modularization*. They are described as follows:

- *Maintainability* represents the student's ability to write durable and adaptable code to new needs.
- *Cyclomatic Complexity* informs the complexity of a programming code that is the number of paths of a method [Curtis et al. 1979].
- *Indentation metric* characterizes the instructions of a program within structures and functions.
- *Laconism* expresses the capacity to express itself in a few words that in programming is measured by the number of tokens per line of code.
- *Modularization* informs organizational capacity of the parts of a functional or data module.

Then, bringing together the cognitive matrices for each programming solution of a course, a *3D Representation of Learning Profiles* of a programming class. The same procedure is performed for a *Reduced Matrix*. This timeline formed by a set of *Learning States* of a student over a course is called *Learning Profile*.

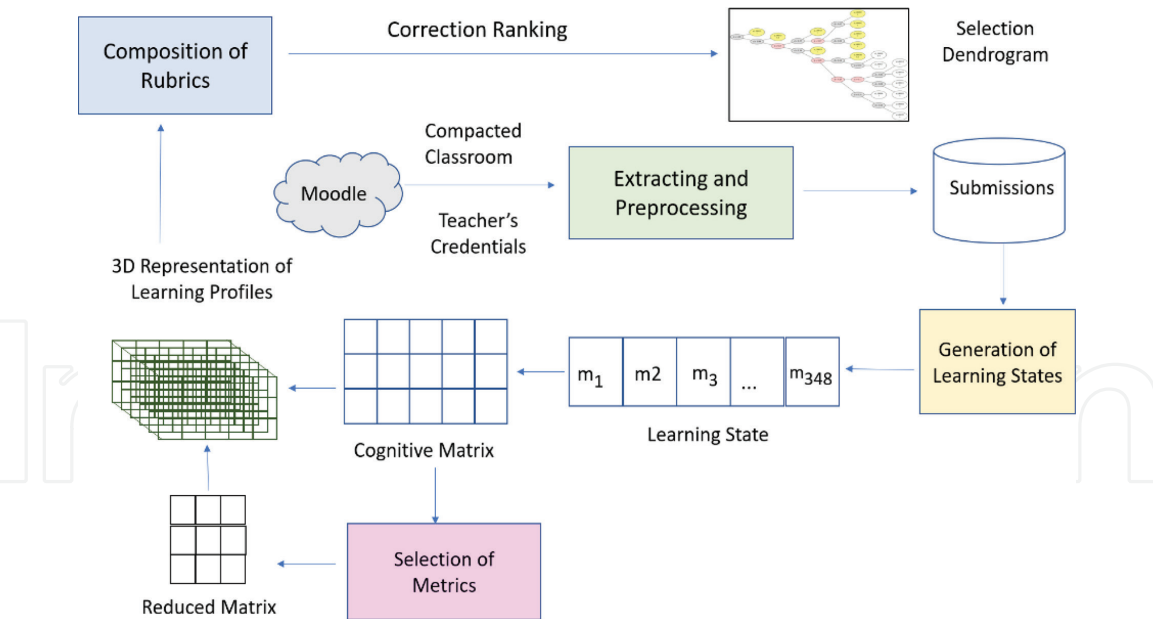


Figure 1.
Architecture of the 3D representation system of students' profiles.

Learning Profile shows how a set of student's assessment variables evolves over a course. Thus, through the analysis of profiles of learning it is possible to understand main learning difficulties of students and to reorient teaching with formative assessment actions in order to anticipate the predictable future of poor performances.

3.1 Selection of metrics

The Reduced Matrix generation process is performed by *Selection of Metrics* module (see **Figure 1**) using the *Recursive Feature Elimination* (RFE) method of the *Scikit-Learn* library [19] and a linear regression algorithm. The inputs of *Selection of Metrics* are the grades of some programming solutions and the *Cognitive Matrix* mapped on 348 software metrics generated by *PCodigo II* [3]. The *Selection of Metrics* returns the metrics most related to the grading pattern through a metric ranking.

3.2 Timeline of programming solutions

The timeline consists of a vector representation of the five fundamental metrics or selected metrics most closely related to a teacher's grade from each course programming exercise. This representation contributes to the analysis of how the evaluation metrics evolve for each student during a course and to generate a training set from to predict future exercise performance from history of exercises and performances associated with them.

3.3 Clusters analysis and composition of rubrics

A hierarchical approach we have used s to form clusters of similar solutions. In this way, a representative would be selected from each of these clusters to receive a teacher's grade and that grade would be reproduced for the other standards in the same cluster.

Unlike *PCodigo II* [3], in which clustering is performed with a previously defined number of clusters, a dendrogram based on centroid was generated, from which can be extracted the amount of clusters required, which, in this work, was

placed as half of the samples from the algorithm BFS (*Breadth-first search*) tree, where the depth is given by the distance noted on each edge, that is *Euclidean Distance*.

According to **Figure 1**, for *Composition of Rubrics* with the purpose of assisting teacher's programming exercises, we have developed an automatic selection of representative samples of codes and metrics more related to the marks assigned by a teacher to this small set of representative codes.

In order to select this small set of representative codes, we have used a hierarchical representation of clusters by *Selection Dendrogram* with Euclidean Distance similarity measure. In *Selection Dendrogram*, the first samples marked yellow are the samples selected from *Correction Ranking*, that is a list automatically generated to indicate the best correction sequence of programming exercises so that the teacher can score a smaller set of samples of programs that represents the diversity of marks.

Through this representation, a search in depth not aware of plagiarism is performed starting with the more atypical samples and accumulating distances (from root to node) which are expressed at each node of the dendrogram. Then, after the selected samples are scored by a teacher and the metrics that most impact the grades assigned by him are verified to analyze possible correction inconsistencies.

3.4 Prediction of performance

In order to begin the performance prediction experiments, we have chosen two prediction methods: based on cluster analysis and based on previous performance histories.

In prediction based on cluster analysis, we used the selection ranking that selects representative samples of the *Dendrogram Selection* subgraphs to form a training set of the prediction model based on linear regression with 50% examples of a set of punctuated programming solutions by a teacher. The other 50% are predicted automatically by prediction model with reference based on diversity of the scores assigned by a teacher to the training set examples.

In the prediction of performances based on a history of previous performances, through a time series generated from the 3D representation (students \times activities \times metrics), a regressor model of each metric is trained and a regressor of metrics for grades, then the metrics of the next exercise are predicted as well as your grade. In this case, the training set is represented by the solutions solved by the same student along the course and the note to be predicted is the next solution to the history samples of that set.

4. Experiments and results

The first experiments of the system functionalities proposed in this chapter were in a Moodle's classroom of a distance course of C Programming Language in Brazil. Through the access credentials of a programming teacher, we obtained a zipped copy of the classroom from this course to the processing of learning analysis from students' codes. Next, all C programming code files were extracted along the programming distance course by about 25 programming students.

After the generation of 3D representations of learning profiles (*Activities \times Students \times Metrics*), gathering 10 activities, 25 students and 348 metrics of software, we use this information to generate the following results and views:

- For each activity, the list of metrics that were considered the most relevant to assign marks.

- For a class as a whole, a list of selected metrics from 348 metrics, that were considered the most relevant to assign marks.
- Dendrogram automatically generated on all the metrics that make up the student profile.
- Dendrogram automatically generated on all the metrics that make up the student profile, after normalization to values between 0 and 1.
- A heat map for each activity with selected metrics that best represent each activity.
- A heat map for each activity with the metrics that best represent the marking criterion for the class as a whole.
- A heat map for each student (historical in time) with the metrics that best represent the correction criterion for the class as a whole.
- A heat map for each activity with five metrics representing skills and difficulties programming.
- A heat map for each student (historical in time) with five metrics representing the students' programming skills and difficulties.
- Prediction of student grades, where grades are assigned to submissions that are similar to each other.

One of the activities we use for this experiment was applied in a C programming distance course and contains the following statement:

Write a program to get the number of P points of three teams in a football championship, according to the following mathematical expression:

$$P = 5GP - GN + 3VF + 2VC + E$$

In this formula, GP is the number of positive goals, GN is the number of goals taken, VF is the number of wins away from home, VC is the number of victories at home and E is the number of draws. The output of this program must show, according to the number of points obtained by a team, the champion and the runner-up of a championship.

We chose this activity for learning analysis because the use of logical expressions and conditional and repetitive control structures allows us to differentiate the solutions in order to recognize which solutions show difficulties to construct logical expressions in control structures. In this way, a good solution of this activity will present few comparisons and a few lines of programming code. On the other hand, a solution with several comparisons, instructions and control structures built into the arrangement evidences programming effort and difficulties to construct logical sentences.

In **Figure 2**, using this activity as an example of results 1 and 2, and views 5, 6, we highlight two modes of analysis of programming solutions of our system for a programming activity: first, from software metrics *Maintainability*, *Cyclomatic Complexity*, *Indentation*, *Laconism* and *Modularization* and from metrics that were considered the most relevant for the attribution of marks, that is, the *Reduced Matrix* metrics.

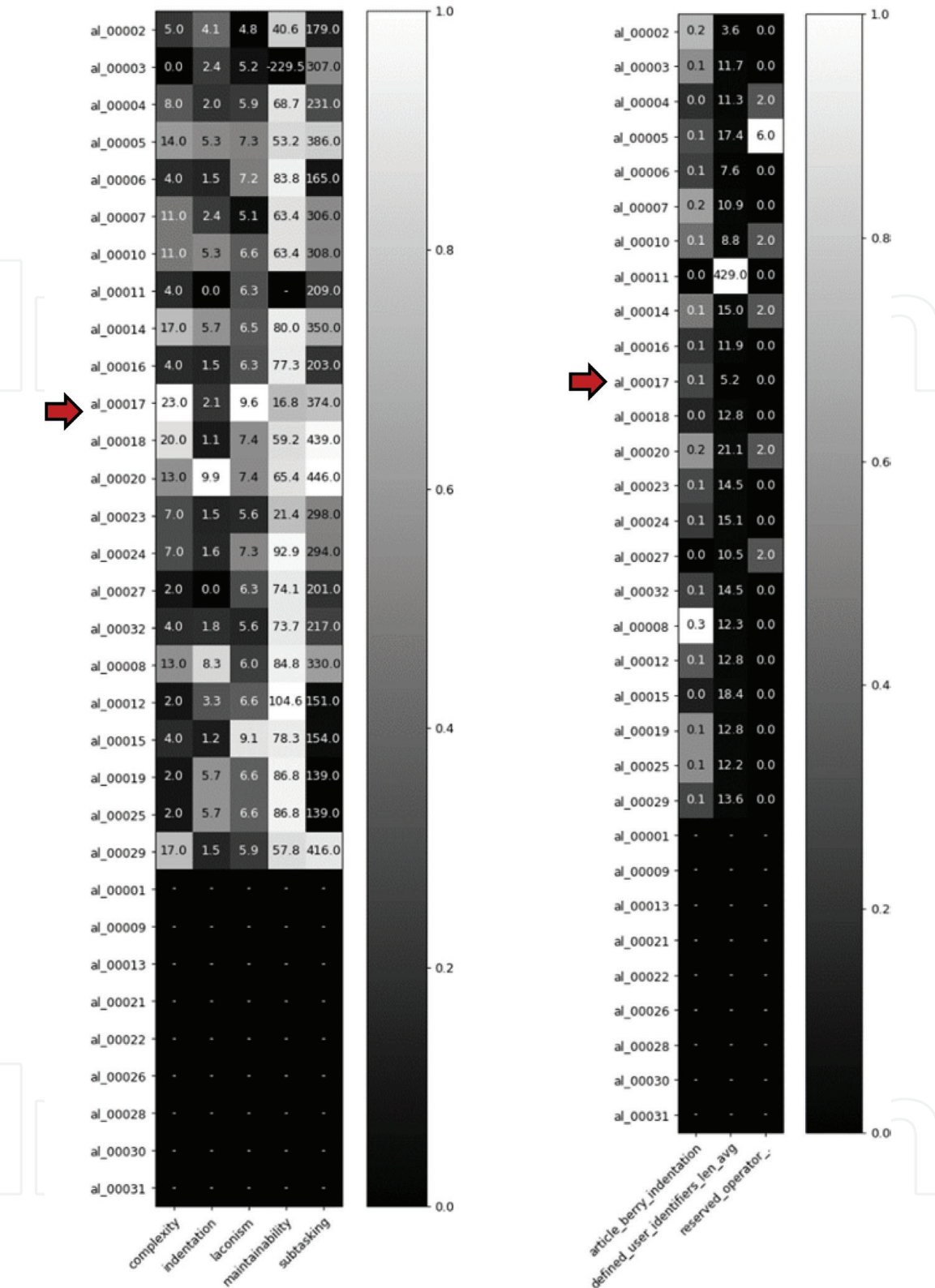


Figure 2.
Analysis of solutions by software metrics.

In graphs of **Figure 2**, the columns indicate students' solutions and columns, metrics. In each column, the white color (scale 1) indicates the highest value and the black color (scale 0), the smallest value of a metric. The interpretation of whether the higher value is better depends on the level of information of each code. However, the teacher can perform this interpretation comparing value of the best solutions and the worst solution. In this way, it would have an instrument to

evaluate which indicators characterize good programming solutions and those that express the most difficulties.

According to **Figure 2**, in the first graph, high value of *Complexity*, low value of *Indentation* and high value of *Laconism* differentiate *al_00017* solution, that is indicated by the red arrow in **Figure 2**, from too much and stand out as a poor solution. In the second graph, however, according to the assessment criteria based on three metrics related to a teacher's mark, this solution follows the pattern of the others and is therefore not indicated as a bad solution.

In **Figure 3**, where there is an example of view 9, we highlight how the five major metrics evolve each exercise for a same student over a course. It is observed that this student, indicated in the first line of the graph by a green arrow, has a predominance of the black color in his programming solution, indicating low values, and meaning good performances in the easiest exercises. On the other hand, in the last exercise by a red arrow he did, the colors appear lighter, indicating more complex activities and more difficulties. That more evident when, from this exercise that has higher *Complexity* value, the student stopped delivering the activities of programming, as it is noticed in the black color indicating a lack of performance in the following activities. We see in this visualization the potential of the tool to enable a teacher to recognize where a student began to demonstrate difficulties.

The graph of **Figure 4** is the ranking view for a teacher to assign marks to activities with the least effort of correcting. This graph is a dendrogram that presents the hierarchy of developed solutions for a programming activity represented by software metrics normalized to values between 0 and 1. Distances are marked in gray and pink.

The graph of **Figure 5** is a ranking view for a teacher to assign marks to activities with the least effort of correcting. This graph is a dendrogram that presents the hierarchy of developed solutions for a programming activity. Distances are marked in gray and pink, and the selected samples are marked in yellow.

According to **Figure 5**, first selecting the samples of greater dissimilarity, the teacher punctuates the most different ones and then some of the more similar ones.

As this teacher follows the ranking of samples suggested by the system, he himself can identify how far he can correct to obtain a minimum set of representation of the diversity of the solutions developed for composition of rubrics and, in the future, for to train automatic assessment exercises of programming exercises with a set of examples of teachers' marks. In this case, we consider 50% for training and 50% for testing of the prediction model.

Figure 6 presents our first prediction results performed at a distance learning C programming. In this graph, we present performance results of all the programming solutions developed by a student (*al_00009*) throughout a programming course. In the presentation of these results, for each submitted programming solution, we compared the grade given by a teacher with the grades predicted by our system from a history of activities previously solved by that same student and from solutions of other students of class in that same activity based on nearest neighbor methods. This process of performance analysis is performed for all students of the distance learning course through our system.

According to the graph of **Figure 6**, it is observed that the prediction of a student's performance in an activity based on a history of exercises solved by that student and in the solutions of that exercise developed by other students still present themselves divergent from the assigned marks by a teacher, although in higher performances these approaches approximate the evaluation of a teacher.

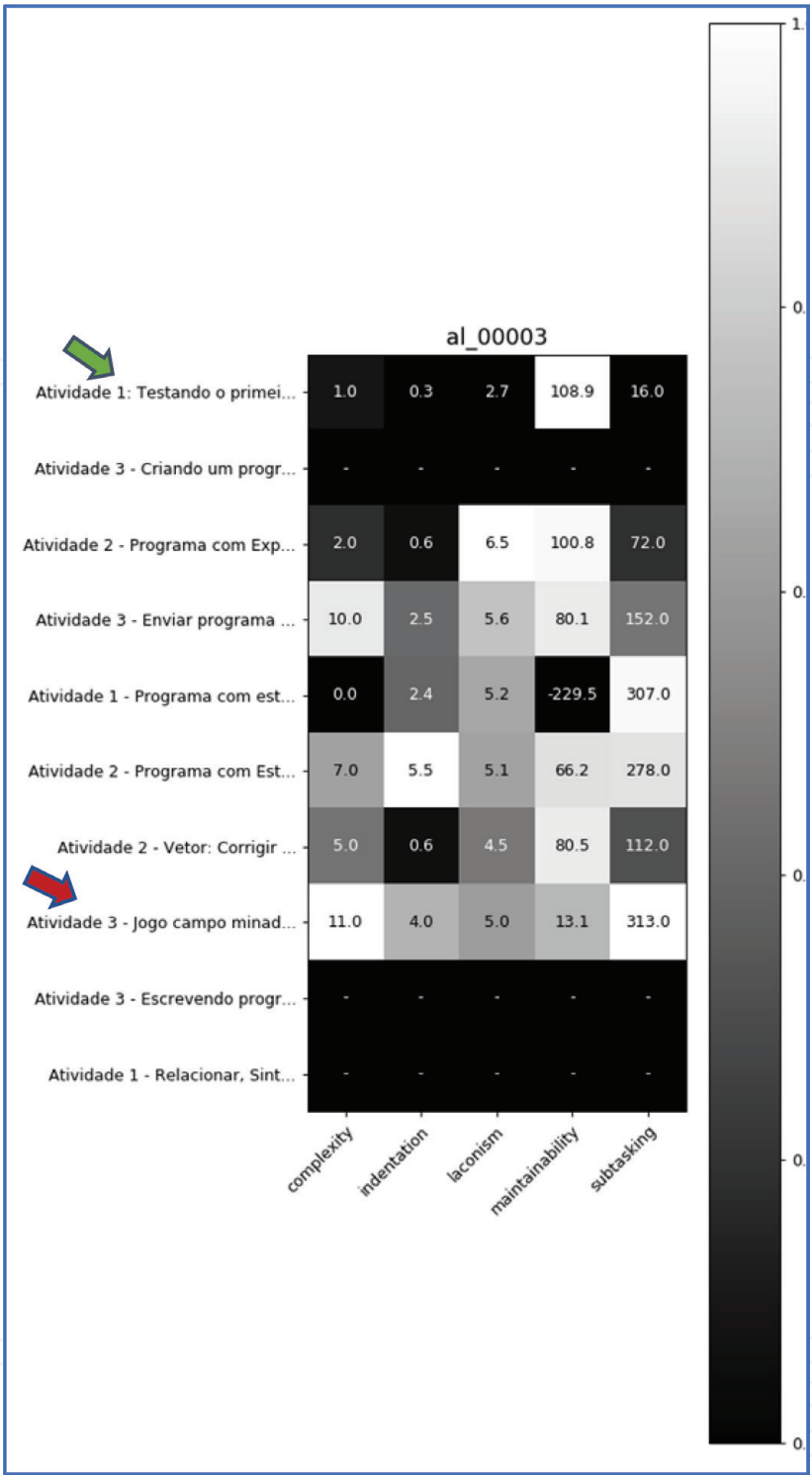


Figure 3.
Evolution of metrics for each activity.

In addition, the predictive results of these approaches approximate as the history of solved exercises used increases. Thus, we present good expectations to advance in the studies of these methods to predict the performance of programming students.

In conclusion, with some examples of the results generated by the system of this chapter, we shown the potential of this tool for programming teachers to accompany the process of learning their students from the beginning to the end of a course from a broad or reduced set of metrics and with less teachers' evaluation effort.

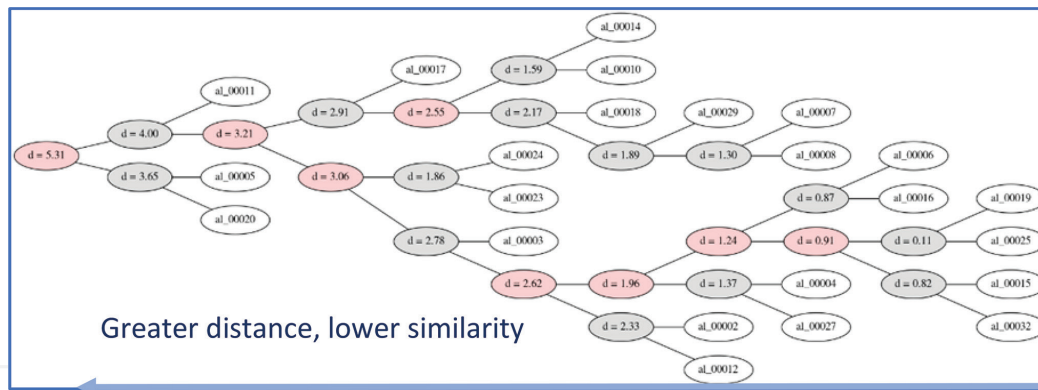


Figure 4. Dendrogram of solutions of a programming activity represented on normalized software metrics (without grades).

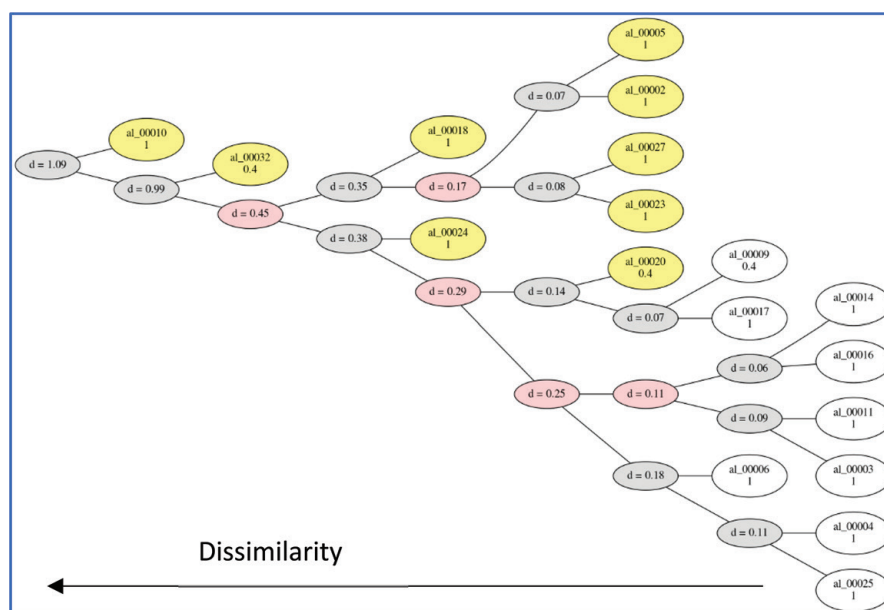


Figure 5.
Dendrogram of solutions of a programming activity selected from a correction ranking (with grades).

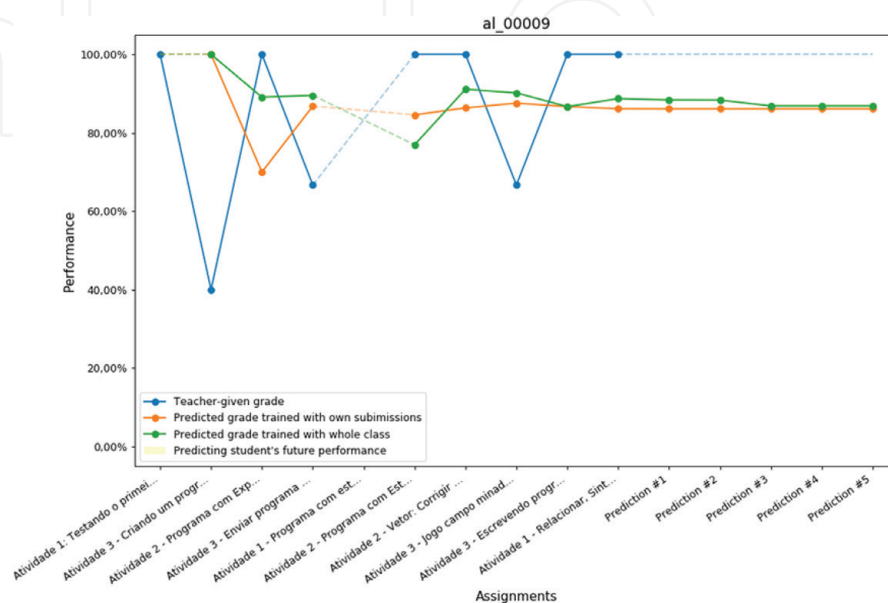


Figure 6.
Timeline with prediction of performance in programming.

5. Conclusion

The system proposed in this chapter was presented as a relevant tool to assist teachers in their evaluation decisions, enabling them to assist the learning process of their students in each programming exercise.

For this, our system can recognize where the learning difficulties begin, monitor how students evolve along a course, generate rubric representation and, soon, predict future performances of programming students.

These possibilities of learning analysis contribute a lot to reducing teachers' efforts in the onerous task of evaluating programming exercises so that they can better track the learning process of students and reorient their formative actions.

Some future works from this research are using samples indicated for manual correction as training references of a semi-automatic programming evaluation system and improving our strategy to predict performances in activities from the timeline of solved programming exercises or from students' solutions that solved exercises similar to the one we intend to predict a grade.


Through this work we offer, therefore, a multidimensional and the clinical analysis tool to help teachers in their formative assessment actions and students to be better assisted in their difficulties and skills in the practice of programming.

Author details

Márcia Gonçalves de Oliveira*, Ádler Oliveira Silva Neves
and Mônica Ferreira Silva Lopes
Federal Institute of Espírito Santo, Vitória, Espírito Santo, Brazil

*Address all correspondence to: marcia.oliveira@ifes.edu.br

IntechOpen

© 2018 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] De Oliveira MG, Marques Ciarelli P, Oliveira E. Recommendation of programming activities by multi-label classification for a formative assessment of students. *Expert Systems With Applications*. 2013;**40**(16):6641-6651
- [2] Ferreira Novais D, Varanda Pereira MJ, Rangel Henriques P. Profile detection through source code static analysis. *Drops-Idn/6014*. 2016;**51**(9):1-9
- [3] Neves A, Reblin L, França H, Lopes M, Oliveira M, Oliveira E. Mapeamento Automático de Perfis de Estudantes em Métricas de Software para Análise de Aprendizagem de Programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. 2017;**28**(1):1337
- [4] Curtis B, Sheppard SB, Milliman P, Borst MA, Love T. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. *IEEE Transactions on Software Engineering*. 1979;**5**(2):96-104
- [5] Berry RE, Meekings BAE. A style analysis of C programs. *Communications of the ACM*. 1985;**28**:80-88
- [6] Khirulnuzam AR, Ahmad S, Nordin J. The Design of an Automated C Programming Assessment Using Pseudo-code Comparison Technique. *National Conference on Software Engineering and Computer Systems*. 2007;1-10
- [7] Xu S, Chee YS. Transformation-based diagnosis of student programs for programming tutoring systems. *IEEE Transactions on Software Engineering*. 2003;**29**:360-384
- [8] Naude KA, Greyling JH, Vogts D. Marking student programs using graph similarity. *Computers in Education*. 2010;**54**(2):545-561
- [9] Rees MJ. Automatic assessment aids for Pascal programs. *SIGPLAN Notices*. 1982;**17**(10):33-42
- [10] Hung S, Kwok L, Chung A. New metrics for automated programming assessment. In: *Proceedings of the IFIP WG34/SEARCC (SRIG on Education and Training) Working Conference on Software Engineering Education*. Amsterdam, The Netherlands: North-Holland Publishing Co.; 1993. pp. 233-243
- [11] de Oliveira MG, Reblin LL, de Souza MB, Oliveira E. Automatic recognition of rubric representations in programming exercises clusters. *Brazilian Journal of Computers in Education*. 2018;**26**(02):60
- [12] Kumar V, Boulanger D, Seanosky J, Panneerselvam K, Somasundaram TS, et al. Competence analytics. *Journal of Computers in Education*. 2014;**1**(4):251-270
- [13] Halstead MH. *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier Science Inc.; 1977
- [14] Blikstein P, Worsley M, Piech C, Sahami M, Cooper S, Koller D. Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming. *The Journal of the Learning Sciences*. 2014;**23**(4):561-599
- [15] Oliveira MG, Monroy NAJ, Zandonade E, Oliveira E. Análise de componentes latentes da aprendizagem

de programação para mapeamento e classificação de perfis. In Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2014;25(1):134-143

[16] Oliveira M, França H, Neves A, Lopes M, Silva AC. Instrumentos de Visualização da Informação para Avaliação Diagnóstica em Curso de Programação a Distância. In: Anais do Workshop de Informática na Escola. October 2017;23(1):452

[17] Spalenza M, Oliveira E, Oliveira M, Nogueira M. Uso de Mapa de Características na Avaliação de Textos Curtos nos Ambientes Virtuais de Aprendizagem. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2016. p. 1165. Available from: <http://www.br-ie.org/pub/index.php/sbie/article/view/6802>

[18] Tang H, Xing W, Pei B. Time really matters: Understanding the temporal dimension of online learning using educational data mining. Journal of Educational Computing Research:0735633118784705. Available from: https://journals.sagepub.com/doi/pdf/10.1177/0735633118784705?casa_token=gYB8xtamE-AAAAAA:byyb5nlAyEnPrkI8u7gAtJNjn5Il4hysSOTAmSGBB1DLTCkjPJ3kqYm8Qy7iFTo3AHSfa59mDuAK5Q

[19] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in python. Journal of Machine Learning Research. 2011;12:2825-2830. Available from: <http://dl.acm.org/citation.cfm?id=2078195%5Cnhttp://arxiv.org/abs/1201.0490>