Combined Methodology for Theoretical Computing

Gabriel Spadon de Souza*, Pedro Henrique de Andrade Gomes†, Ronaldo Celso Messias Correia†,
Celso Olivete Júnior†, Danilo Medeiros Eler† and Rogério Eduardo Garcia†

*Departamento de Ciências de Computação, Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo (USP) – São Carlos, São Paulo, Brazil
†Departamento de Matemática e Computação, Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista (UNESP) – Presidente Prudente, São Paulo, Brazil

spadon@usp.br, {gomes, ronaldo, olivete, daniloeler, rogerio}@fct.unesp.br

***Abstract*—Theoretical Computer Science area (TCS) stands out by being an important study field, and it is composed by Formal Languages and Automata Theory (FLA), Computer Science Theory (CST), and Theory of Compilers (TC). This area is responsible for introducing the beginnings of the Computer Science through formalisms – which represent a set of methods, techniques, or rules that describe the solution to a problem with restrictions – and it has a substantial impact on the student's knowledge. Computer science theory is based on the understanding of computability and techniques to solve challenges, and to improve the teaching-learning process used to introduce these concepts we proposed a Combined Methodology for Theoretical Computing (CMTC). Our methodology is based on formalism development to ground the knowledge acquired during classes of FLA, CST, and TC, where students are introduced to Theoretical Computing during one year and a half. In each course, we applied the same methodology where each student used data structures, computer graphics, and algorithms to solve problems. We address this methodology to understand how much the incomprehension of formalisms is influenced by new concepts and its abstractions. Against this background, we demonstrate that the that CMTC has the aim to build knowledge and make the new concepts and formalisms concrete. Our results are based on statistical analysis from students' grades, where we could observe among other results, the correlation between the practical activities and the conceptual knowledge.**

I. Introduction

Theoretical Computer Science area (TCS) is a valuable study field, and it is composed by Formal Languages and Automata Theory (FLA), Computer Science Theory (CST), and Theory of Compilers (TC). This area is responsible for introducing the beginnings of the Computer Science through formalisms[1], which has a substantial impact on the students' knowledge.

A common way to teach formalisms is to use sketches to make a visual representation of an abstract concept. However, draw sketches take too much time, and by being a static image, they do not help with the interaction between students and formalisms. For this reason, many simulation tools were proposed to support the teaching-learning process, and all of them were used to support and to reproduce a formalism through visual illustration [1].

[1]A formalism represents a method, technique or a set of rules that describe a way to solve a problem considering its restrictions.

Guided by computability and techniques to solve problems, TCS classes require focusing on Formal and Regular Languages; Regular and Context-Free Grammars; Nondeterministic and Deterministic Finite Automaton; Finite State; Post, Moore and Turing Machine; and others. By the high heterogeneity, volume, and complexity of information about formalisms, its concepts can be reinforced when the student learns how to transform a formalism in its programming abstractions. Therefore, simulation tools generate an interactive model from an abstraction and help students to understand the logic through visual representation. On the other hand, the simulation is not enough to describe the formalism development. Consequently, the teaching-learning process needs to be supported by e-tools, which are capable of interaction, enhancing the learning logic and demonstrating how to develop a formalism with programming languages and its paradigms.

Several tools have been developed to introduce formalisms structure [1]–[5] and to help the learning process, contributing to build a concrete knowledge and to minimize difficulties faced by students [6]. However, they are not enough to overcome those difficulties, because the different perspectives of teaching and developing require novel methodological approaches to support the teaching-learning process.

To improve the teaching-learning process, we proposed a Combined Methodology for Theoretical Computing (CMTC), based on simulator development to ground the knowledge acquired in the TCS area during classes of FLA, CST, and TC. The experience of modeling a simulator provides the opportunity to the student transform their perception in a code architecture [5]. As a result, this approach provides challenges and make connections to different representations which have a common abstraction. With this strategy, students can develop their formalisms as a method to clarify the course topics.

In our methodology, students are introduced to Theoretical Computing during one year and a half. In each course, we applied the same methodology where each student used data structures, computer graphics, and algorithms to solve problems. Against this background, we demonstrate that the that CMTC has the aim to build knowledge and make the new concepts and formalisms concrete. Our results are based

on statistical analysis of the students' grades, where we could observe among other results, the correlation between the practical activities and the conceptual knowledge.

This paper is divided into five sections, apart from the Introduction it is organized as follows: Section II presents some related works; Section III presents the Combined Methodology for Theoretical Computing and its evaluation methods; Sections IV presents the statistical analysis of the students' grades; Section V presents the final remarks and future works.

II. Related Works

Several studies were addressed by the literature seeking for better educational tools, self-learning methods and evaluation techniques. Regarding the need for the continuous improvement in the teaching-learning process, the former studies focused on achieving better results in knowledge exchanging between teachers and students.

Constructivism is an important educational method, and it is defined as a learning theory described by psychology, which explains how people might acquire information and learn from it. Ben-Ari [7] discussed this approach and its impacts on the learning process, as well as Papert and Harel [8]. The constructivist approach considers the self-learning as the most important step to improve the student's knowledge, in another perspective, it can be seen as a way to acquire knowledge from knowledge itself.

The use of learning tools it is not an exclusive method, many e-tools and even some educational models were introduced by Ribeiro *et al.* [9], Cardim *et al.* [10], Messias *et al.* [5] and Husain *et al.* [11]. All of them discussed structured approaches to teach data structure and algorithms, considering the technological support in the pedagogical process.

On the other hand, it is widely debated the use of theoretical computer simulators for Formal Languages and Automata Theory (FLA) [12], [13], as well as its teaching-learning methodologies [1], [14], in particular, the problem-based learning method [15], apart from the failure in comprehending Theoretical Computing classes [16]. However, none of them saw the learning process as a single set of courses that together can enhance the formalism understanding by a combined methodology.

In the following, we briefly introduce some of the simulators earlier introduced in the literature and which were used as e-tools in our classes.

Formalism Simulators

jFAST is a Java finite automata simulator, which focuses on visualization and interactivity to provide active learning techniques in order to improve the understanding of abstract concepts. It categorizes itself as an easy-to-use tool for teachers and students, with an emphasis on finite state machines. The aim of this simulator is to enhance the teaching effectiveness in FLA courses, particularly for less advanced computer science students. However, the simulator is limited to the finite automata representation [2].

JFLAP is a theoretical computer science framework. It was built to design and simulate several variations of automata and computing machines. It is the most embracing tool developed in this area, and it is capable of several conversions from one representation to another [17].

Visual **A**utomata **S**imulator (**VAS**) is a tool capable of simulating and transform computational formalisms in programming abstractions. It is limited to finite automata and Turing machines [18].

Auger is a simulator that can visually represent finite automata and convert it to a regular grammar, and it is limited by this representation, which, in turn, describe the lowest level of complexity on Chomsky hierarchy [19].

Automatograph is a deterministic finite automata simulator and is a beta software. It has been used to help the teaching-learning process with regular grammars and finite state automata. The aim of this application is to validate characters' chains with an automaton dynamically created, showing the steps of recognition until the formalism reaches the final state [4].

Deus **E**x **M**achina (**DEM**) is a simulator capable of providing a generic platform for designing and running different kinds of automata, it stands out by implementing Markov algorithms. It was built on an icon-based interface where students can sketch an automaton using nodes and arcs. It solves problems step-by-step showing how the formalism work in a given input string [13].

As we can see, there are many tools capable of representing the formalisms, and usually, the authors show the tools focusing on a single feature available. They spent much time creating an icon-based interface to interact with the student and explaining how the formalism works without introducing a way to build it. Moreover, tools are not tied to a methodology, and an adequate methodological approach is required to improve the knowledge exchange, which can be, not exclusively, supported by e-tools.

III. Methodological Approach

TCS is introduced to the students using the Chomsky hierarchy [20], consisting of a containment hierarchy of classes of formal grammars. In a course, it is divided into topics where each one explains and presents a formalism with the fundamental theory and its implementation guidelines.

Formal Languages and Automata Theory (FLA) and Computer Science Theory (CST) aim to provide the ways for understanding procedures and computability methods, apart from improving the students' ability to solve complex problems. Those courses are offered once a year, and they have a workload of 60 hours each, where half of it is reserved for teaching the theoretical concepts and the other half to practice through practical exercises. The main bibliography used is based on the work of Aho and Hopcroft [21], Aho, Alfred and Sethi [22], Menezes [23], Diverio and Menezes [24], Hopcroft, Ullman and Motwani [25], and Prince and Toscani [26].

Theory of Compilers (TC) has the purpose of teaching the compilation process and its stages. The course introduces

the main concepts and development techniques, enabling the student to build a compiler for a prototype language with a subset of commands and instructions. The course is offered once a year and they have a workload of 60 hours, like the others, half of the workload is reserved for teaching the theoretical concepts and the other half to practice through the development of a compiler. The bibliography used to conduct this course is the same to both FLA and CST classes being the only difference between the courses is how deeply the concepts are taught.

Each course topic is taught individually, and we defined an order to teach it by its complexity; thus, we can teach each formalism from its raw to complex form. Our classes present each one of them in the same order that they are described, as listed in the following:

1) Regular and context-free languages, and their grammar;
2) Deterministic and non-deterministic finite automata;
3) Push-down automata and phrase structure;
4) Turing, among other machines;
5) Linear-bounded machine and context-sensitive grammars;
6) Halting problem and propositions on computability;
7) Thesis of Church-Turing;
8) Concepts of compilation and interpretation;
9) Structure of a compiler;
10) Lexical, syntactic (forward and reverse) and semantic analysis;
11) Table of symbols and error management;
12) Intermediate code, code generation, and optimization concepts.

Based on these previously introduced topics, our methodology consists of evaluating the students through its theoretical and practical knowledge, where the practical knowledge is the result of the development of a simulator capable of representing formalisms taught in class. As a novel method, we explored the impact of the practical knowledge on the TCS courses, for that, we connect different courses enabling the continuous development of the formalism simulator. The motivation of our work is the related class-topics which allow us to explore a single one with far more advanced features than those in former works, in addition to testing new teaching strategies and contributing to the robustness of knowledge.

To help the teaching process and to introduce theoretical concepts, we have used learning tools capable of generating visual representations from formalisms. As presented in Section II, the literature introduces many educational tools focused on some of the presented learning topics. However, to motivate our students to build their simulator, we used tools developed by other students in previous years. Specifically, we used the tool presented by Souza *et al.* [1] in addition to a web-based compiler.

During the classes, the students arranged themselves in groups, and they were challenged to build a formalism simulator based on steps previously defined by the professor. The construction of the practical knowledge was made by proposing the development of a new module on the students' simulator, which was made for each class-topic as soon as the topic was taught. For that, we followed the concepts presented by Messias *et al.* [5], where the authors state that the students need to be taught about the theory to represent and solve problems with each formalism. Thus, to evaluate the practical work and quantify the acquired practical knowledge for each student we analyze each simulator focusing on its conceptual implementation and its ability to solve problems. Consequently, the higher grades of a simulator do not represent a simulator that successfully implements all formalisms and give us correct results for any test case, but the understanding to create a data structure capable to represent its ideas.

In this sense, we quantify our achievements through the students' grades, where the minimum grade required to be approved is 05.00 of 10.00 points. The students were evaluated through one theoretical exam on each trimester (eq. 1) and the simulator evaluation at the end of each semester (eq. 2). The final grade is the average of all evaluations made during the semester if both grades (theoretical and practical exam) are greater than or equal to 5.00; otherwise, the final grade is the lowest between the two (eq. 3).

$$T^2 = \frac{t1 + t2}{2}; \tag{1}$$

$$S^3 = \frac{topic[0] + ... + topic[n]}{n - 1}; \tag{2}$$

$$F^4 = \begin{cases} \frac{T+S}{2}, & \text{if } T \geq 5.0 \; and \; S \geq 5.0; \\ lowest(T, S), & \text{otherwise} \end{cases} \tag{3}$$

We do not expect the students' grade always stays near to 10.00. Rather, it is desired to show that over the years, with our approach, more students can be approved. Also, we intend to show that there is a correlation between the test score and the simulator development. As a result, the higher number of approved students is a result of the teaching-learning methodology, which is capable of making the students more focused and stimulated during the classes.

There is evidence that our methodology facilitates the understanding of abstractions and the formalism construction. The simulator's development supports the teaching-learning methodology presented in this paper, and it has a fundamental importance for Computer Science students, especially those who aspires the Theoretical Computing as a professional or academical career.

In order to verify the results of our methodology, in the next section, we introduce the analytic results from the students' grades. We collected the grades for all students who attended the disciplines in its logical order (FLA, CST, and TC), and on these data, we study the correlation between the test score and the simulator score.

IV. Results

Five years ago a professor of our department started to question the methods to introduce complex and raw topics to Computer Science students. It is a well-known fact that this issue has been preoccupying the experts for some time, and a

[2]T denotes the mean grade of tests scores.
[3]S denotes the mean grade of the simulator score.
[4]F denotes the final grade of the student.

great deal is being written and said about it (see Section II). However, to answer this question, we begin by taking a closer look at the students' difficulties and not to the learned topics, and with an empirical approach, we repeatedly observed their lack of logical abstraction. Based on our observation, we developed a couple of methods to introduce the raw topics to the students and the one which stands out is the simulator development.

Five years later, to test our methodology, we collected test and practical work scores. Inside this time window we have taught fifteen different classes, each class had an average of twenty-five students excluding repeaters. Our final dataset consists of a student's unique identifier, followed by its test score and practical work score, ordered by the year the student attended the course.

Over these five years, in three of them, we applied our methodology based on simulator's development as a combined methodology on TCS courses. Thus, we extracted this data from our dataset using a sub-window of three years. Besides, to analyze this information we adopted two approaches: on the first one, to check its behavior, we have analyzed the grades without considering the course order (using the five-years data); on the second one, to quantify the learning improvement, we sorted the dataset considering the courses order and removing non-intermittent classes (using the three-years data).

A. Primary approach

At first, we plot the students' id and his/her final grades; the grades were divided into courses so we could compare one score against the others for the same student. This result is depicted in the first subfigure at Figure 1.

As a first conclusion, we observe that on TC classes, students have achieved lower scores compared to the other classes. Then, reviewing our data and our evaluation method (eq. 3), we conclude that these students, in particular, not developed the simulator, consequently, even with good test grades the final grade is the lowest one. The question here is why these students do not achieve good grades, and when asked, the most common answer is the lack of time caused to the high workload.

On the other hand, in the third subfigure depicted in Figure 1, we can observe the high variability of the TC's grades when compared to the other courses. Thus, if we compare FLA with TC we can statistically infer that the TC's scores are better than FLA, and we consider that is a result of the increased learning logic achieved by the students. We noticed that with the first contact with highly abstract concepts, students have adapted and accomplished better grades. Nevertheless, the students' adaptation did not increase the class average score significantly.

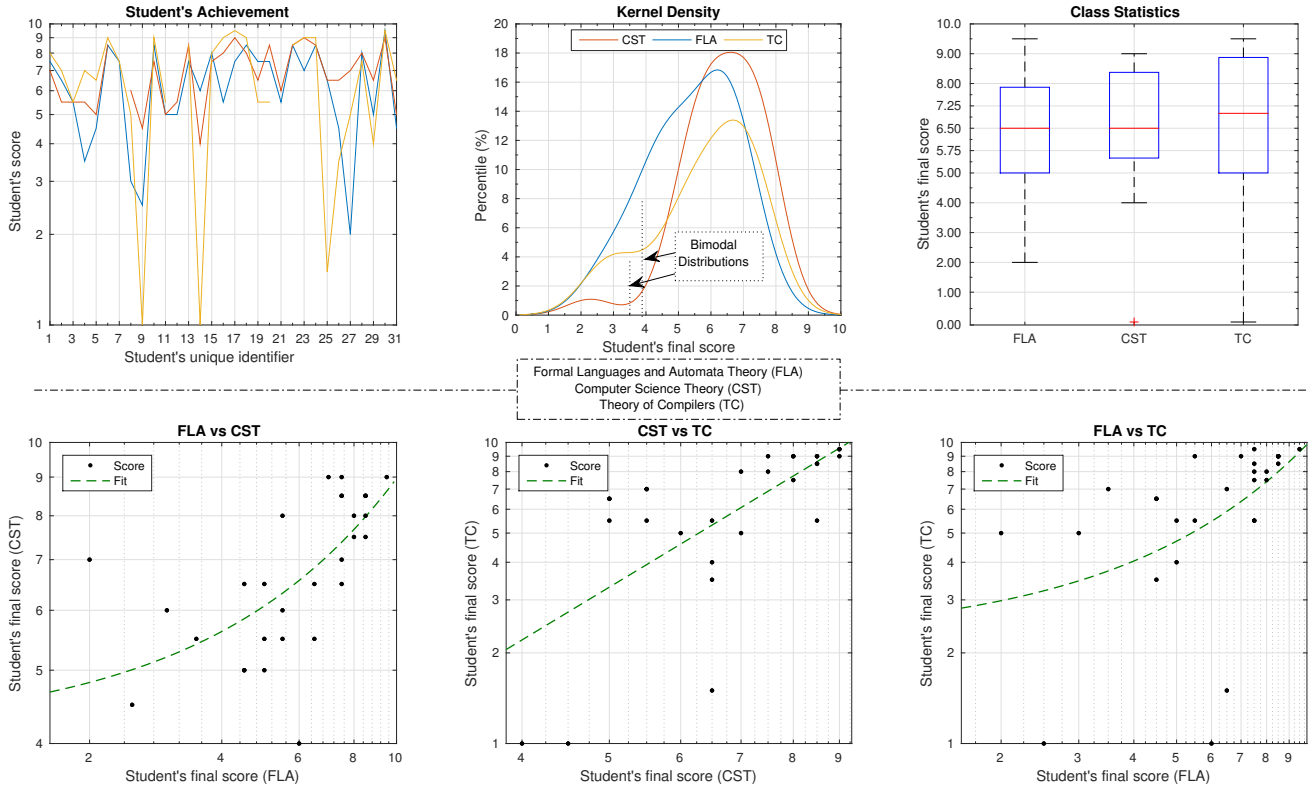

Fig. 1. Statistical evaluation of the students' grades. At the top of the image is depicted the preliminary analysis of the grades, where the subplot at left shows the final score variation for a small set of students; the subplot in the middle shows the Kernel Density results from the final grades; and, the subplot at right illustrate its variance, median, and outliers as a boxplot. Besides, at the bottom of the image is depicted dependency analysis of the courses, where we show a supra-linear tendency from a course to another.

Another point is shown by the second subfigure depicted in Figure 1, which reveals the percentile of students with the same score. To accomplish it, we used the Kernel Density to estimate the probability density function of the final grades. The data would seem to suggest that most of its population has a score greater than five, denoting high approval rate in all courses. Besides, a noteworthy result is the Gaussian bimodal behavior on the distribution of CST and TC characterizing the outliers of our data.

B. Secondary approach

To check the learning improvement achieved with our methodology, we found in the correlation analysis a powerful set of measures to validate our methods, which was firstly noticed with empirical tests and exploration. In this approach, we make use of Pearson Correlation and k-Means to identify k clusters in our dataset.

First of all, without applying any specific method, we plot the data in pairs one against the others (subfigures in the bottom of Figure 1), which represent a final grade map, showing in x-axis one course and on y-axis another. Thus, we fit the data with the best model considering its coordinates on the Euclidean space. The fit shows us a supra-linear (polynomial) tendency, where the score achieved on the current course impacts the score on next course – considering FLA, CST, and TC, respectively. This is evidence that there is a correlation between variables in our data. Consequently, we analyzed a single course at the time, measuring the correlation of the simulator compared to the test score.

To do so we used *Pearson Correlation Coefficient*, which is capable of evaluating the correlation degree of two linear variables, it is denoted by ρ and has its values in the range of $[-1.0, +1.0]$ [27]. For this measure $\rho = 1.0$ denotes a perfect linear positive correlation, where two variables increase positively in the same proportion; on the other hand, $\rho = -1.0$ represents a perfect linear negative correlation, where the variables oppositely increase in the same proportion; despite the fact that $\rho = 0.0$ indicates the existence of no linear correlation between the two variables.

Using Pearson, we evaluate the correlation coefficient for each pair of axis on the Figure 2 and we divided it into two categories: "3-Years" and "1-Year". The results are presented in Table I.

In the "3-Years" category, where we took into account three years of the use of our methodology, we achieved a positive correlation $\rho \geq 0.8$ for all courses. This result shows us that our methodology has an enormous impact on the final grades of the students and mainly on simulator's scores.

If we consider the Kernel Density results, we can show that the most of our students have a final score greater than 05.00 points. Then, if we combine it with the correlation coefficient of the grades, knowing that the theoretical concepts are materialized as a simulator, we can see the substantial impact of our methodology on the final grades, once the simulator is a tool that develops the concepts evaluated in the test.

Besides, in the "1-Year" category, we present the class with less correlated students' grades. This interval corresponds to the first year where our methodology was used and at this time it was still in development, so we consider that this result reflects the adaptation time of the students and teachers. Despite this fact, we still can see the positive correlation in students' grades, and this shows us that even in a short period the proposed methodology was able to achieve good results.

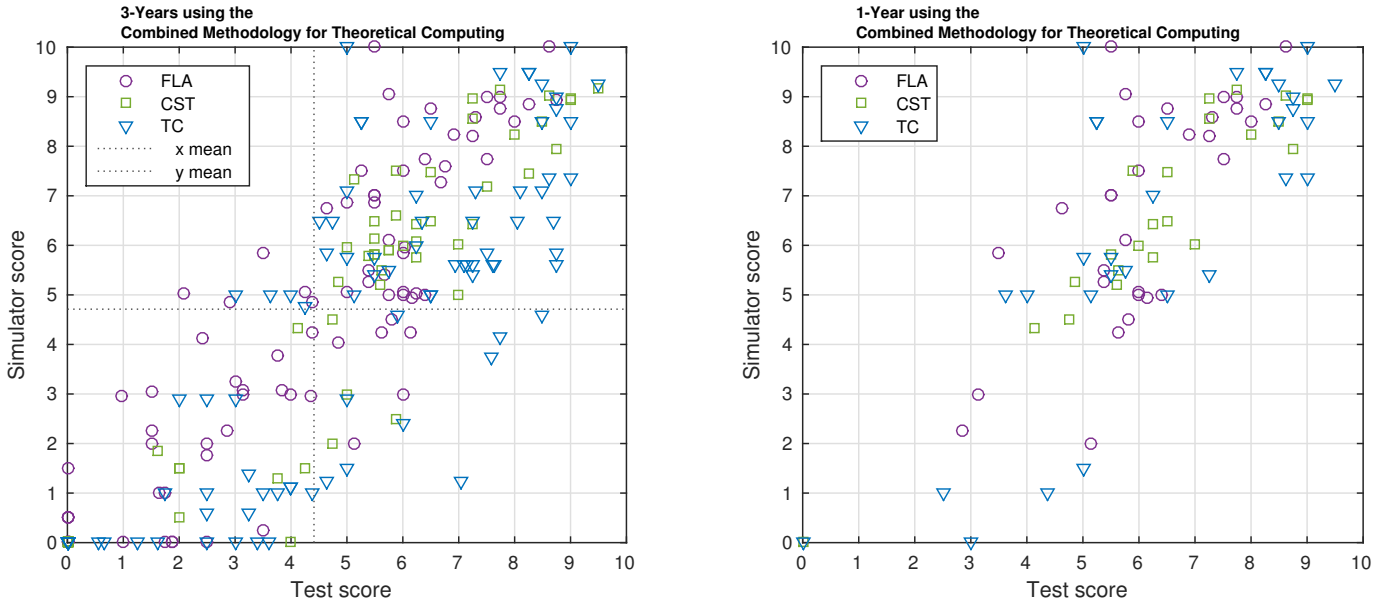

Fig. 2. Scatter graphs from the students' grades showing the simulator score depending on the test score and vice versa. The subplot at left represents the grades of the students during three years where we applied our methodology; at right, the subplot shows the grades of the year with less correlation inside the three years' interval. For each subplot, the grades were separated in its courses (FLA, CST, and TC).

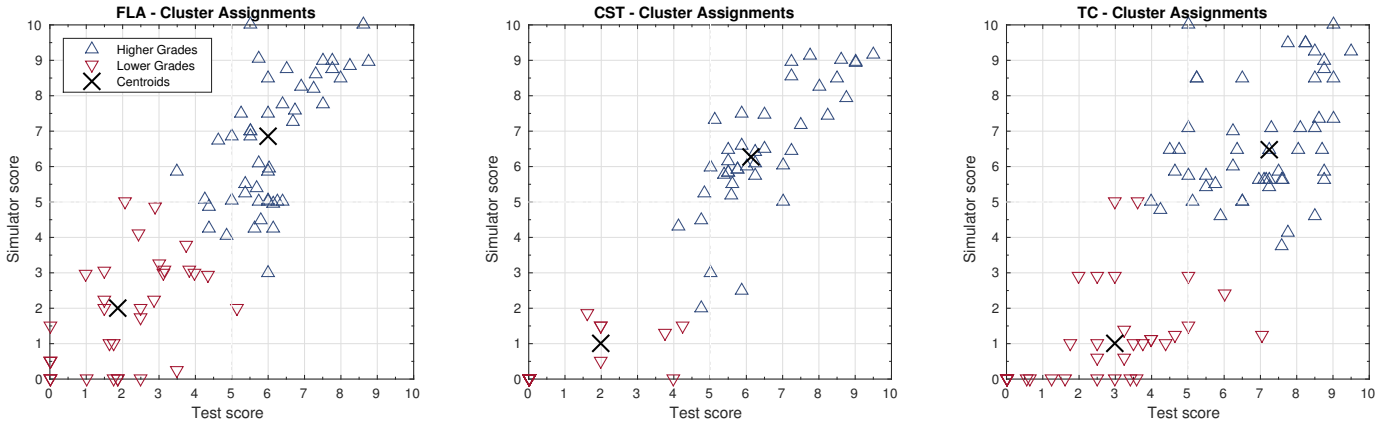

Fig. 3. Scatter graphs from the students' grades showing the simulator score depending on the test score and vice versa; the final grades were split into its courses (FLA, CST, and TC) and it was identified the cluster with higher and lower grades, apart from its centroids. The subplot at left represents the results from the FLA courses, in the middle we present the CST courses, and at right, the subplot shows the grades from the TC courses.

Pearson Correlation			
	1-Year	3-Years	Correlation Type
FLA	0,7053	0,8467	linear positive
CST	0,9425	0,9073	linear positive
TC	0,7905	0,8054	linear positive

Table I. Correlation coefficient of the simulator and the test grade for each student who attend our classes; the 3-Years column represents the time window where we applied our methodology; the 1-Year column represents the year with less correlation; and Type column shows the type of correlation found.

As the last analysis, we used k-Means with $k = 2$ (Figure 3) to group the students in two different clusters. Our purpose was to split the grades in one cluster with the higher ones and the other with the lower ones. For each cluster we found its centroid – the centroid is an equilibrium point inside a cluster, and it can be seen as the average of all points of the cluster – the centroid is the expected score considering all other students in the same cluster.

The cluster analysis is not an immutable science when we split the grades it does not mean that the higher grades will be always greater than 05.00, and the lower grades will always be lower than it. So, if we make an imaginary cut in the middle of each axis, we can split it into four quadrants, respectively delimited in the intervals of $[0, 5[$ and $[5, 10]$, and we will get a better analysis based on each quadrant.

Based on this result, we can easily see that the most grades are in the $\{(x, y)|x \in [5, 10] \ and \ y \in [5, 10]\}$ or $\{(x, y)|x \in [0, 5[\ and \ y \in [0, 5[\}$, showing an expected behavior. Besides, can be seen that is unlikable to have outliers populating the other quadrants – outliers, in this case, represent isolated situations apart from the methodology, for an example, students who have lost many classes. We can see then, that the centroids, as an equilibrium point for each cluster, denotes the expected grade from each element on it.

The centroid tends to attract the items in the cluster, and that is clearly seen on the CST course, where the outliers have its centroid in $[2, 1]$ for the lower grade cluster. On the other hand, the higher grade cluster on TC is highly sparse, almost at the border of the quadrant, indicating high variability and maybe an inaccurate centroid.

V. Conclusion

In this paper, we present a Combined Methodology for Theoretical Computing (CMTC), based on simulator development to ground the knowledge acquired in the TCS area during classes of Formal Languages and Automata Theory (FLA), Computer Science Theory (CST) and Theory of Compilers (TC). According to the methodology, students are introduced to Theoretical Computing (TC) during one year and a half. In each course, we applied the same methodology where each student used data structures, computer graphics, and algorithms to solve problems. During the classes, the students arranged themselves in groups, and they were challenged to build a formalism simulator for each class topic. In our methodology, the simulator development represents the incremental construction of the students' practical knowledge which is capable of guiding the student to the formalism understanding.

We used the developed methodology during the teaching process at the "Universidade Estadual Paulista (UNESP)", over five years. Our evaluation consisted of using the students' grades divided in simulator's score and test score. We decided to evaluate the methodology because it has a strong influence on how the students understand a formalism.

We have noticed that the CMTC, seeking for the knowledge construction, allow students to make the concept concrete. Thus, regarding the discussed results, presented using Kernel Density, Cluster analysis, and Person Correlation Coefficient, we confirm the theoretical concepts taught in the courses when materialized as a simulator have a substantial impact on the final grades. Also, according to the teacher, the students have shown more interest and more motivation during the classes.

This methodology was slightly modified and is currently being tested on other courses from different areas, and until know, it helped students to expand their knowledge on classes of distributed systems and databases, these students, in particular, have reached an academic level of knowledge, allowing

specific scientific contributions as presented by Souza *et al.* [28], Santana *el al.* [29] and Souza *et al.* [30].

Finally, to enable the replication of our methodology we provide on a website[5] three simulators used to present the course topics to the students. Each simulator was developed in the previous year, and they are focused on FLA, CST and TC classes.

ACKNOWLEDGMENT

We are grateful to CNPq – "National Counsel of Technological and Scientific Development" by the support (processes 9254601/M and 457875/2014-3) and to the Department of Mathematics and Computer Science (DMC) at Faculty of Science and Technology (FCT) by the resources granted to this research.

REFERENCES

[1] G. S. de Souza, C. Olivete, R. C. M. Correia, and R. E. Garcia, "Teaching-learning methodology for formal languages and automata theory," in *Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE*. IEEE, 2015, pp. 1–7.

[2] T. M. White and T. P. Way, "jfast: A java finite automata simulator," in *ACM SIGCSE Bulletin*, vol. 38, no. 1. ACM, 2006, pp. 384–388.

[3] S. H. Rodger and T. W. Finley, *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[4] E. B. Kienetz, "Desenvolvimento de uma ferramenta para auxílio na aprendizagem de linguagens formais–automatograph," *Salão de iniciação Científica (18.: 2006: Porto Alegre, RS). Livro de resumos. Porto Alegre: UFRGS, 2006.*, 2006.

[5] R. C. Messias Correia, R. E. Garcia, C. Olivete, A. Costacurta Brandi, and G. P. Cardim, "A methodological approach to use technological support on teaching and learning data structures," in *Frontiers in Education Conference (FIE), 2014 IEEE*. IEEE, 2014, pp. 1–8.

[6] B. J. Ferreira, "Em busca de uma prática pedagógica inclusiva: Uma experiência no ensino de estruturas de dados mediada por animações gráficas," in *XV Workshop sobre Educação em Informática. Anais do XXVII Congresso da SBC. Rio de Janeiro*, 2007.

[7] M. Ben-Ari, "Constructivism in computer science education," in *Acm sigcse bulletin*, vol. 30, no. 1. ACM, 1998, pp. 257–261.

[8] S. Papert and I. Harel, "Situating constructionism," *Constructionism*, vol. 36, pp. 1–11, 1991.

[9] R. da Silva Ribeiro, L. d. O. Brandão, and A. A. Brandão, "Uma visão do cenário nacional do ensino de algoritmos e programação: uma proposta baseada no paradigma de programação visual," in *Anais do Simpósio Brasileiro de Informática na Educação*, vol. 23, no. 1, 2012.

[10] G. P. Cardim, I. Marçal, C. M. De Sousa, D. L. De Campos, C. H. Marin, A. F. Do Carmo, D. F. Toledo, A. Saito, R. C. M. Correia, and R. E. Garcia, "Teaching and learning data structures supported by computers: An experiment using cadilag tool," in *Iberian Conference on Information Systems and Technologies, CISTI*, 2012.

[11] M. Husain, S. Patil, B. Indira, S. Meena, and D. Narayan, "Structured approach of designing data structure and algorithms laboratory experiments," *Journal of Engineering Education Transformations*, vol. 29, no. 2, pp. 83–88, 2015.

[12] M. Procopiuc, O. Procopiuc, and S. H. Rodger, "Visualization and interaction in the computer science formal languages course with jflap," in *fie*. IEEE, 1996, pp. 121–125.

[5]http://www2.fct.unesp.br/docentes/dmec/olivete/

[13] C. I. Chesñevar, M. L. Cobo, and W. Yurcik, "Using theoretical computer simulators for formal languages and automata theory," *ACM SIGCSE Bulletin*, vol. 35, no. 2, pp. 33–37, 2003.

[14] C. I. Chesnevar, M. P. González, and A. G. Maguitman, "Didactic strategies for promoting significant learning in formal languages and automata theory," in *ACM SIGCSE Bulletin*, vol. 36, no. 3. ACM, 2004, pp. 7–11.

[15] W. Hämäläinen, "Problem-based learning of theoretical computer science," in *Frontiers in Education, 2004. FIE 2004. 34th Annual*. IEEE, 2004, pp. S1H–1.

[16] M. A. Qureshi, M. F. Hassan, and R. Sammi, "Rationale of students' failure in comprehending theoretical computer science courses," in *Computer and Information Sciences (ICCOINS), 2014 International Conference on*. IEEE, 2014, pp. 1–5.

[17] S. H. Rodger, H. Qin, and J. Su, "Changes to jflap to increase its use in courses," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. ACM, 2011, pp. 339–339.

[18] J. Bovet, "Visual automata simulator," 2004.

[19] C. Szymanski and L. A. S. Garcindo, "Auger–ferramenta de apoio ao ensino de autômatos finitos," *Unisul, Santa Catarina–SC*, 2004.

[20] N. Chomsky, "On certain formal properties of grammars," *Information and control*, vol. 2, no. 2, pp. 137–167, 1959.

[21] A. V. Aho and J. E. Hopcroft, *Design & Analysis of Computer Algorithms*. Pearson Education India, 1974.

[22] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques*. Addison wesley, 1986.

[23] P. B. Menezes, *Linguagens formais e autômatos*. Sagra-Dcluzzato, 1998.

[24] T. A. Diverio and P. B. Menezes, *Teoria da Computação: Máquinas Universais e Computabilidade-Vol. 5*. Bookman Editora, 2009.

[25] J. E. Hopcroft and J. D. Ullman, "Formal languages and their relation to automata," 1969.

[26] A. M. d. A. Price and S. S. Toscani, *Implementação de linguagens de programação: compiladores*. Sagra-Luzzatto, 2000.

[27] D. LeBlanc, *Statistics: Concepts and Applications for Science*, ser. Statistics: Concepts and Applications for Science. Jones and Bartlett, 2004, no. v. 2.

[28] G. S. Souza, R. C. Messias Correia, R. E. Garcia, and C. Olivete, "Simulation and analysis applied on virtualization to build hadoop clusters," in *Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on*. IEEE, 2015, pp. 1–7.

[29] V. J. Santana, G. Spadon de Souza, R. C. Messias Correia, R. E. Garcia, D. Medeiros Eler, and C. Olivete, "Scalable information system using event oriented programming and nosql," in *Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on*. IEEE, 2015, pp. 1–6.

[30] G. S. Souza, R. C. Messias Correia, R. E. Garcia, C. Olivete, and B. R. G. Santos, "Health information system for medical survey analysis," in *Information Systems and Technologies (CISTI), 2016 11th Iberian Conference on*. IEEE, 2016, pp. 324–330.