# Static Analysis Model For Assessing Source Codes With TFIDF

1st Ricardo Lemos de Souza
*PPG Modelagem Computacional*
*Universidade Federal do Rio Grande*
Rio Grande - RS, Brazil
rcrdsou@hotmail.com

2nd Fabiana Zaffalon Ferreira
*PPG Educação em Ciências*
*Universidade Federal do Rio Grande*
Rio Grande - RS, Brazil
fabinhazaffalon@gmail.com

3rd Silvia da Silva Botelho
*Centro de Ciências Computacionais- C3*
*Universidade Federal do Rio Grande)*
Rio Grande - RS, Brazil
silviacb.botelho@gmail.com

***Abstract*—This Full Paper in the category Research-to-Practice presents a proposed model for assessing source codes through static analysis, an applied experiment and results. The presence of computation is constantly growing in the contemporaneous world, and in that way, the demand for professionals capable to develop and maintain software is also in constant growth. The present work aims to present and discuss the results obtained by applying the proposed model based in TFIDF to a dataset. Results showed that by enabling the comparison of different skills present on each source code, the proposed model offers an asset for teachers to identify potential weaknesses on a student´s set of computer programming skills, and therefore be able to work on solutions for their educational development.**

***Index Terms*—Static analysis, source code assessment, computer programming education, TFIDF**

## I. Introduction

The presence of computation grows in the most different aspects of contemporaneity. Whether on computers, smart phones, vehicles or appliances capable of connecting to the internet, there is an increasing number of objects that have some type of embedded software [1], [2]. The greater presence of computing in all sectors also leads to an increase in the demand for professionals capable of developing and maintaining computational systems.

Introductory programming courses are known to have a high number of failure and dropout rates [3], [4]. Learning computer programming encompasses the development of a set of skills such as logic, mathematics, as well as others not directly related to computer science [4], [5]. Newcomers to programming face not only the challenge of learning a programming language, but also how to apply their previous knowledge to an algorithmic solution [5]. The difficulty of relating different skills for this purpose is presented as one of the main justifications for the levels of failure and dropout [5], [6].

In face-to-face courses, exercises for implementing algorithms in source codes are widely used in programming disciplines and, with the increase in the number of students, the number of codes resulting from the activities carried out by students can generate a high demand for educators, as a result, it is common to search for a model of automatic evaluation of source codes. In order to meet the growing demand for programmers, there is also the spread of new programming teaching environments, such as courses on online platforms, which constantly seek to develop methodologies to provide better teaching, as well as more accurate evaluation methodologies [7], [8].

Web courses and platforms offer the educator the opportunity to concentrate administration, organization, and assignment of activities to students. Many educational institutions have developed their own activity management systems for teaching programming, others make use of platforms developed by third parties, such as Online Judges [8]. Online Judges are platforms that have a bank of programming problems. On these platforms the problems are available for users to search, choose and submit a source code with the solution for them. Once the solution is submitted, it is judged, that is, an automatic evaluation system produces and informs the result of its evaluation. As an example of Online Judges, we can mention the platforms URI Online Judge [9] and Codeforces [10].

Automatic assessment methods are common in objective questions, however the universe of possible answers in a programming exercise is similar to essay questions, that is, even if two source codes are different they can reach the same result. The most used evaluation methods in programming are included in two classifications: dynamic and static [7], [11].

Most evaluation systems make use of dynamic analysis of source codes (DA), so that given a given set of inputs, the set of outputs generated by the source code is compared with a set previously determined [11]. This method requires the execution of the source code and is often used as a binary evaluation, determining whether the answer is correct or not.

Static Analysis (SA) of source codes, does not require the execution of the evaluated code, uses the quantification of elements present in the source code as the number of identifiers and operators, and is able to evaluate even solutions that could not be compiled and performed due to some error [7].

It can be said that there are still other evaluation models, such as hybrid and semi-automatic. The hybrid evaluation contemplates aspects of both dynamic and static evaluation. The semi-automatic evaluation makes use of the automatic evaluation in some aspects, but it does not exclude the need

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

for human evaluation for others. In an educational context, it is possible to identify many benefits of automatic assessment, such as speed, consistency, objectivity, and availability. However, in this same educational context, automatic assessment systems can generate feedback that does not satisfy the subjectivity necessary for monitoring the student [7].

This work aims to present an experiment that makes use of a proposed model with TFIDF (Term Frequency – Inverse Term Frequency) adapted to source code static analysis. The aim is to identify groups of elements in the code, which we believe represents a set of skills related to the solution of a given problem. As a reference for programming skills, we made use of Computational Thinking competences that was related to source code structures elsewhere.

## II. Background

Introductory computer programming course are acknowledged by high failure and dropout rates [4] [3]. Novice programmers are face with not only the challenge of learning a computer-based language, but also on how to apply previous knowledge, e.g., mathematics, logic and text interpretation, in a way to elaborate an algorithmic solution to problems [5].

Virtual learning environments (VLE) are widely used for computer programming courses and provide an opportunity for educators to track the development of student's skills by analyzing their source code submissions, and many of this VLE make use of automatic assessment (AA) methodology [11]. Most AA systems make use of either dynamic or static assessment, which are well discussed in the works of Ulla [11] and Ala-Mutka [7]. In this work, static analysis of source codes was chosen as the methodology for assessment, given the ability to examine even codes that fail to be interpreted by compilers.

As for the evaluation in an educational context, it is considered that its main function must be to allow a critical analysis of the educational reality, the discovery of new problems, new needs and possible dimensions [12]. Thus, the evaluation consists of a diagnostic process [13], that is, it aims to identify positive and negative elements of the educational process, enabling an adaptation in the direction of the same.

The assessment is an important step in the educational process, which commonly aims to assess whether the student's performance is adequate to the expected development, that is, it seeks to verify whether the assessed subject demonstrates the desirable skills.

Therefore, when contemplating evaluating a student, it is necessary to have a definition of the concepts of skills and competences. Competences can be understood as the global or practical mastery of an everyday situation, and skills as the mastery of an operation specific training or actions that meet one or more competences [14].

Information retrieval methodologies have been successfully used in the evaluation of source codes. In that way, the use of natural language processing techniques was made, as in the works of Azcona [15], Ulla [16] and Ganguly [17], transforming the formal computer program language in which the codes were submitted to a numeric representation, more adequate to apply statistical techniques.

There is no universal definition for information, but this concept can be simplified as something that has a symbolic representation, a structured organization, and that can be understood by a subject [18]. According to Boyce et al. [18], information retrieval systems are applications, computational or not, that seek to retrieve desired information in a data collection. Information retrieval systems from textual data often use term balancing techniques [19]. Balancing is a process that assigns weights that reflect the different relevance of a term for the retrieval of particular information.

In his work Saracevic [20] presents a review of the evolution of the concept of relevance in information science, this original work was reworked and expanded in [21] and [22]. Fundamentally, relevance is a tuple, understood by the relationship between its different parts: tangible as documents, machines, and processes; intangibles such as ideas, concepts and information; properties such as topicality, utility and specificity [21].

There are different methodologies used to assign weights, we can distinguish them as being manual or automatic. Initially the information sciences used manual assignments for specificity, such as a Boolean assignment whether the term is relevant or not, or even rankings of the terms. This type of interpretation is based on a semantic interpretation by an appraiser, and is considered an onerous, subjective, and imprecise task [18].

In his work, Jones [23] proposed a shift from interpreting the specificity of a term to its relevance in a data set. For this specificity, previously estimated through an attribution of semantic relevance, the author proposed the change to a statistical measure given by a function of term using, this paper is then recognized as the first reference to the TF-IDF.

TF-IDF is given by the product of the term frequency (TF - Document Frequency), presented in (1) and the inverse frequency of the document (IDF - Inverse Document Frequency), presented in (2), where $f$ represents the frequency of a term in the document, $T$ the total number of terms in the document, $N$ the total number of documents in the collection and n the total number of documents in which the term is present [24]

$$TF = \frac{f}{T} \tag{1}$$

$$IDF = log\left(\frac{N}{n}\right) \tag{2}$$

What are the computer programming skills and competences? There is no consensus in the researched literature, nevertheless, there is some prominent set of competences to a given context, for education one of the most prominent set of competences are those of the Computational Thinking [25]. Computational thinking (CT) deals with the set of skills related to a problem-solving approach analogous to the way a computer operates, i.e., the thinking process involved in formulating a problem and in expression of your solution so

that a computer - human or machine - can effectively carry out [25].

In their research Barr and Stephenson [26] carry out a study of how to insert computational thinking skills in the K-12 curriculum. In this process, with the help of specialists, it relates the competences of computational thinking with different areas of the curriculum, including computer science, in which the listed capacities or activities involve structures or elements that can be observed in a source code such as the use of repetition, conditional commands and definition of functions.

Next it is presented the methodology applied for identifying and estimating competences through the analysis of source codes.

## III. Methodology

To assess competences in source codes, it is required to define the set o competences to be used as reference. As presented in the previous sections, there is no consensus in the available literature about what are the computer skill, or competences. Therefore, for the present work it was chosen one prominent set of competences used in computer programming evaluation, the Computational Thinking competences. More specifically, it was used as reference the Computational Thinking competences described by Barr and Stephenson [26].

The authors of the present work acknowledges that the chosen competences are not the unique possible reference for programming skills, or even exclusive to this goal. Defining which competences are used in computer programming is not the goal of the present work, neither to establish which are the computational thinking competences, but, given a set of competences, to verify if it is possible to quantify and estimate competences levels through the analysis of source codes.

Even though Barr and Stephenson [26] have described competence elements that are possible to identify in source codes, in order to be able to numerically represent those structures it was required to define and relate then to a specific formal computer programming language. The 'GNU C' language was chosen for this purpose in the present work. Using the cited reference [26] initial concepts, expanding those concepts to examples for other areas of knowledge also present in referred work when the understanding of a broader context was required, also expanding those descriptions to other works discussed in the previous sections, the following set of competences were defined for the present work:

1) Data collection: Data collection was described in the work chosen as reference as: "find a data source for a problem". It is understood that this ability corresponds to the structures of data entry in a source code. Thus, the defined parameters are tokens related to obtaining external data. Observing the grammar of the C language, it was parametized the structures commonly used for data entry in programming exercises, present in the library "stdio.h" and "mysql.h".
2) Data analysis Data analysis was described in the work chosen as reference as: "Write a program to do basic statistical calculations on a set of data". It is understood as tokens that are used to perform arithmetic operations. Thus, the defined parameters are given by the presence of arithmetic operators of the C grammar.
3) Data representation: Data representation was described in the work chosen as reference as: "Use data structures such as array, linked list, stack, queue, graph, hash table, etc." It is understood as structures used in programming to represent information or a set of information.
4) Problem decomposition: Problem decomposition was described in the work chosen as reference as: "Define objects and methods; define main and functions". It is understood as structures that aim to break down the general problem into smaller ones. In this sense, the declaration of functions or procedures and the use of macros were identified
5) Abstraction: Abstraction was described in the work chosen as reference as: "Use procedures to encapsulate a set of often repeated commands that perform a function; use conditionals, loops, recursion, etc." It is understood as structures that represent central relations in a piece of source code. It took as a direct reference the occurrence of logical and relational operators used in conditions, repetition commands and decision structures.
6) Simulation: Simulation was described in the work chosen as reference as: "Algorithm animation, parameter sweeping". It is understood that the concept is related to tokens that are used for debugging, testing and animation.

Other concepts cited on the reference work have been excluded from the present work for either not being on the scope of this paper, or not being adequate for the 'GNU C' language. Algorithm and procedures concept was excluded for being related to the research of classic algorithms, which is not the goal ins this work. The Concept Automation was excluded for, elaborating a algorithm as a source code is automation in its essence. Finally, Concept Parallelization was excluded for not being present in the original C language concept, only available through C11 standards. Table I presents the chosen competences and related tokens.

TABLE I
Skill concept and tokens

| Competence | Tokens |
|---|---|
| Data Collection | Scanf, DB calls, File Calls |
| Data Analysis | Arithmetic operator |
| Data representation | Identifiers , Arrays, Pointers, Structs |
| Problem Decomposition | Function Definitions, Source imports, Macro definitions |
| Abstraction | Function calls, Logic operators, Loops, recursive calls, Selections |
| Simulation | Comments, printf, plots |

The aim of this study is to analyze data sets, so that each source code processed individually makes up a larger set. For the present work, it was determined as the set of solutions to the same problem. For each problem, it is proposed to build

three matrices: a document matrix by token, a document by frequency matrix, and a balanced matrix.

The document by token matrix is formed by the set of solutions where each row corresponds to a submission, and each column corresponds to the tokens extracted in their sequential form. This matrix corresponds to the raw data, processed in order to transform the original code into a numerical representation.

The document by frequency matrix is formed by the set of solutions where each row corresponds to a submission, and each column corresponds to the frequency of the tokens related to each of the skills, according to table I. To build the frequency of the skills, the number of occurrences for each token is counted and then those related to the same skill are added, that is, the frequency of a skill is the sum of all occurrences of the tokens related to it.

The frequency matrix is then balanced using the TFIDF method, according to (3), (4), (5), where $f$ represents the frequency, $t$ term or token, d document or submission and $N$ the total number of submissions. The balanced information is then stored in another matrix, a balanced document by TFIDF matrix.

$$tf_{t,d} = \frac{f_{t,d}}{max\left(f_{t,d} : t \in d\right)} \tag{3}$$

$$idf_{t,D} = log\frac{N}{\{d \in D : t \in d\}} \tag{4}$$

$$tfidf_{t,d,D} = tf_{t,d} \times idf_{t,D} \tag{5}$$

It has been observed in preliminary experiments that all competences were present in all valid source codes analyzed, although at different levels. Applying the proposed groupings of tokens by competence it was already expected that no competence frequency would be equal to zero, in this case (4) would obtain the result zero ($log(1) = 0$), making the method unfeasible. In order to adjust the use of the IDF, only the frequency of documents in which the token count is greater than the sum of the mean $m_{t,d}$ with its standard deviation $std_{t,D}$ was considered, thus (4) is balanced as (6).

$$idf_{t,D} = log\frac{N}{\{d \in D : t > (m_{t,D} + std_{t,D}) \in d\}} \tag{6}$$

The use of TF-IDF in the present work aims to emerge the importance of a skill for building a solution from a source code. The average importance of a skill to the set of solutions is also observed. However, it is understood that the TF-IDF does not apply to the comparison of the quality of the skills present in different solutions. Next, we seek to attend this purpose.

For the present work, a comparison system between codes that compose a set of answers for the same exercise was used as a guide to infer the quality of the skills used in the development of source codes.

Given the characteristic of the present work, in which the information is extracted from the frequency of tokens and used to estimate competences, given a certain set of solutions to the same problem, it was assumed that either a competency has a quality inversely proportional $\theta I$ to the lowest frequency of tokens observed, indicated by (7), or a quality directly proportional $\theta D$ to the highest frequency of tokens observed, indicated by (8). This assumption is based on the principle that, for inversely proportional competences, if source code $A$ is able to present a solution with a lower frequency of tokens related to competency $C$ compared to code $B$, $A$ probably makes use of the competence $C$ more appropriately. In contrast, for directly proportional competences, if source code $A$ has a higher frequency of tokens related to competency $C$ compared to source code $B$, source $A$ probably makes use of the competence analyzed in a more appropriate way.

$$\theta I_{t,d} = \frac{\min f_{t,d}}{f_t} \tag{7}$$

$$\theta D_{t,d} = \frac{f_t}{\max f_{t,d}} \tag{8}$$

Proportionality was determined as follows:

1) Data collection: inversely proportional
2) Data analysis: inversely proportional.
3) Data representation: inversely proportional.
4) Problem Decomposition: directly proportional.
5) Abstraction: inversely proportional.
6) Simulation: directly proportional.

The TF-IDF aims to emerge the importance of a skill in a submission, the value $\theta$ in turn aims to establish a comparison of the quality of a solution in relation to the other responses referring to the same problem. The TF-IDF determined for each competence was used to balance the calculated values for the Theta proportion.

It is considered that the product of the Proportion $\theta$ and the TF-IDF, represented by $H\theta$ in (9), represents both the importance of a competence for a solution, as well as allowing the comparison of the quality of the competences employed in the elaboration of different solutions, this product was denominated as Theta Skill.

$$H\theta_{t,d} = tfidf_{t,d,D} \times \theta_{t,d} \tag{9}$$

It is considered as $\theta_{t,d}$ in (9) either the result of (7) or the result of (8). The value determined for the Theta Skill for each of the competences represents the estimated quality of each of them, also considering the relevance of each competence for the problem as a whole. For the validation of the proposed model, the following is an experiment in which the proposed methodology is applied to a data set.

## IV. Validation Experiment

For the development of the present experiment, it was chosen to use a database composed of source codes submitted for the solution of six computational problems in an Online Judge platform called Codeforces. Only problems that used the 'GNU C' language and totaled at least 300 submissions

were selected. In this way, 3144 source codes distributed in the six chosen problems were processed.

For data collection, we used the API provided by the Codeforces website itself to generate the list of submissions related to each problem. The information collected from each submission was: submission Id, referring to the identification of the submission, which was omitted in the pre-processing in order to make it impossible to identify the user; problem Id, as the identification of the problem; language, as the formal programming language on which the solution was written. The API made available did not make it possible to obtain the source code of the submissions directly, so a scraper application was developed to collect this information, also directly on the Codeforces website. The source code text information has been incorporated into the corresponding submission as source.

In order to establish a reference for comparing the model's behavior, part of the database was evaluated by two programming teachers. Given the great variability of the types of errors found in codes considered incorrect, such as incomplete codes, sources submitted in an inadequate language, or simply a wrong logic, it was chosen to use only the source codes considered correct for human evaluation, with this we sought to guarantee the homogeneity of the codes functionality, i.e., all evaluated codes are able to solve the problem successfully and differ only in how their solution was elaborated.

The complete set of accepted submissions was provided to the evaluators, together with a description of the competences analyzed and an indication of which structures were considered to be related to each of them. They were asked to assign grades to each competency for each of the source codes in a scale of 1 to 5 points.

It has been necessary to develop a source code analyzer, hereinafter referred to as parser, for the transformation of raw data into numerical representations. For this development effort, GNU Flex software was used for the construction of the lexical analyzer, and GNU Bison for the construction of a syntactic and semantic analyzer. As a reference, the structures to be analyzed are those mentioned in Table I. The objective of the construction of the parser, resulting from the combination of the two analyzers, was to enable the identification of syntactic and semantic structures in the source codes, thus generating a numerical representation of the codes present in the database.

Once the problem representation matrices were defined, the data were processed using RStudio software. For the processing of information, some standard functions of the software were used, such as std(), sum(), mean(), etc., while others necessary for the realization of the experiment were elaborated by the authors.

With the set of submissions transformed into numerical representations and organized into six representation matrices, each corresponding to the set of solutions to a problem, the proposed model was applied to each matrix separately. RStudio software was used for base processing and application of the model, so routines were performed to apply the methodology previously described.

To match the same scale limits used by the human evaluators, the values obtained for the Theta Skill were normalized between values one and five using the method of maximum and minimum according to (10), where $x'$ corresponds to the updated value, $x$ the original value, $a$ the lowest desired value and $b$ the highest desired value.

$$x' = a + \frac{(x - min(x))\,(b - a)}{max(x) - min(x)} \tag{10}$$

It was observed that some of the data obtained did not present a normal behavior, as can be seen in fig 1, which illustrates the Box-Plot graph for the Theta Abilities of problem one. The same behavior of the data was observed in the other problems according to the calculation of the Shapiro-Wilk normality test presented in Table II .

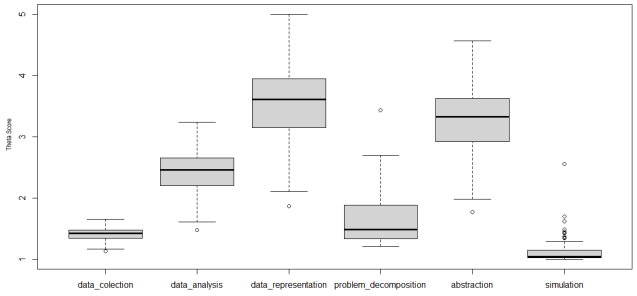



Fig. 1. Box-Plot graphic Theta Skill Problem 1

TABLE II
SHAPIRO-WILK NORMALITY TEST FOR THETA SKILLS

| Problem | Legend | competences | | | | |
|---|---|---|---|---|---|---|
| | | data_analysis | data_representation | problem_decomposition | abstraction | simulation |
| 1 | S-W | 0.986987734 | 0.986987734 | 0.847541117 | 0,986987734 | 0,571882662 |
| | p | 0,555184034 | 0,555184034 | 6,75E-08 | 0,555184034 | 2,32E-14 |
| | h0 | Not rejected | Not rejected | Rejected | Not rejected | Rejected |
| 2 | S-W | 0,872596813 | 0,872596813 | 0,457624202 | 0,872596813 | 0,272310185 |
| | p | 4,65E-11 | 4,65E-11 | 6,25E-23 | 4,65E-11 | 6,51E-26 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| 3 | S-W | 0,966144537 | 0,966144537 | 0,628956821 | 0,966144537 | 0,511700026 |
| | p | 0,000338008 | 0,000338008 | 3,80E-19 | 0,000338008 | 1,13E-21 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| 4 | S-W | 0,971885011 | 0,971885011 | 0,860678009 | 0,971885011 | 0,574425339 |
| | p | 0,029540019 | 0,029540019 | 2,66E-08 | 0,029540019 | 1,20E-15 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| 5 | S-W | 0,985542734 | 0,985542734 | 0,958190118 | 0,985542734 | 0,203871904 |
| | p | 0,314026847 | 0,314026847 | 0,002209977 | 0,314026847 | 2,71E-21 |
| | ho | Not rejected | Not rejected | Rejected | Not rejected | Rejected |
| 6 | S-W | 0,935776885 | 0,935776885 | 0,886600808 | 0,935776885 | 0,566146102 |
| | p | 0,030615362 | 0,030615362 | 0,00108981 | 0,030615362 | 2,06E-09 |
| | ho | Rejected | Rejected | Rejected | Rejected | Rejected |

One of the characteristics that was readily observed with this database is that the structures related to the data collection competence presented a low frequency of tokens. In fact, given the characteristics of Online Judges, in which it is compared if a given set of inputs generate the expected set of outputs, there are almost no variations in the number of data entry structures, the data entry for problems is fixed.

Due to this characteristic, there was an indication that human evaluators may have different perceptions when evaluating a competency; for appraiser 1, the score attributed to

the Data Collection competency in the source codes present in the base was at least two points lower than the rating given by appraiser 2; since, except in rare cases, the number of tokens in these source codes is fixed, it is understood that the difference may be related to the perception of relevance.

In fact, there is no significant variation in the use of the Data Collection competence structures for the elaboration of the source codes present in the database used in this experiment, but there is a significant difference in human evaluations. It can be inferred observing the Data Collection Competence in fig. 1, that this variation does not occur with Theta Skill as it takes into account the dataset as a whole, and the relevance is calculated trough a statistical measure (TFIDF), thus it is not bound to perception. Nevertheless, given the characteristics of the dataset, Concept Data Collection has showed irrelevant in the further analysis of all problems, it is concluded that for this type of source codes, in which entry structures are mostly fixed, the methodology will not suffice any further relevant information, thus, results for Data Collection will not be presented in the analysis to come.

Another observation that can be made from fig 1 is that in general, the Simulation competency does not have a prominent relevance for the elaboration of the solutions for most problems, however, unlike the Data Collection competency, the presence of many outliers is an indication that for some codes the Simulation competency was relevant, differentiating them from the average. The presence of outliers can therefore point to signs that indicate the existence of source codes that are very different from the average, both positively and negatively.

In order to use a reference to analyze the model's behavior, comparisons of data obtained by the model and human evaluations were used. The analyses and comparisons will be made from correlations as an indicative of association and benevolence as comparison of how differently the model grades from the evaluators.

The Kendall non-parametric correlation test was chosen as in a previous study of the data, a non-normal behavior was observed for some competences, according to table II that presents the Shapiro-Wilk test. Not finding a standard for reference in the researched literature, limit values for the Kendall test were established for the correlations as: $(correlation < 0.1) = null$; $(0.1 < correlation < 0.3) = low$; $(0.3 < correlation < 0.5) = average$; $(correlation > 0,5) = strong$.

To calculate the correlations, RStudio's cor.test() function was used, with the kendall parameter. The Tau value is analyzed, which determines the intensity of the correlation, and the p value, which determines the significance of the test for the sample, serving as a reference to indicate the presence of evidence that allows the rejection of the null hypothesis that the data are independent. Table III shows the correlations calculated between the proposed model and the marks attributed by the evaluator 1, while Table IV shows the correlations between the model and the evaluator 2.

TABLE III
CORRELATION THETA X APPRAISER 1

| Problem | Legend | competences | | | | |
|---|---|---|---|---|---|---|
| | | data_analysis | data_representation | problem_decomposition | abstraction | simulation |
| 1 | tau | 0.494942725 | 0.511394145 | 0.575361302 | 0.575368106 | 0.513103117 |
| | p | 1.25E-08 | 1.61E-09 | 6.90E-11 | 1.53E-11 | 7.76E-09 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| | Intensidade | Average | Strong | Strong | Strong | Strong |
| 2 | tau | 0.467609334 | 0.420689033 | 0.250875318 | 0.609428475 | 0.404355601 |
| | p | 7.23E-13 | 1.50E-11 | 0.000114404 | 5.69E-24 | 3.42E-10 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| | | Average | Average | Weak | Strong | Average |
| 3 | tau | 0.603877457 | 0.574565552 | 0.397525917 | 0.618620194 | 0.418457597 |
| | p | 3.22E-23 | 1.37E-21 | 1.70E-10 | 3.44E-25 | 4.77E-11 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| | | Strong | Strong | Average | Strong | Average |
| 4 | tau | 0.456005716 | 0.412374956 | 0.235358998 | 0.575378778 | 0.390601976 |
| | p | 7.75E-09 | 1.03E-07 | 0.004781373 | 1.39E-13 | 2.01E-06 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| | | Average | Average | Weak | Strong | Average |
| 5 | tau | 0.545599537 | 0.386207634 | 0.224623455 | 0.582442936 | 0.419116477 |
| | p | 3.63E-12 | 6.68E-07 | 0.006025687 | 1.49E-14 | 2.46E-07 |
| | ho | Rejected | Rejected | Rejected | Rejected | Rejected |
| | | Strong | Average | Weak | Strong | Average |
| 6 | tau | 0.622694908 | 0.665988402 | 0.480741102 | 0.76359387 | 0.508697392 |
| | p | 1.38E-06 | 1.18E-07 | 0.000166953 | 1.01E-09 | 0.000125646 |
| | ho | Rejected | Rejected | Rejected | Rejected | Rejected |
| | | Strong | Strong | Average | Strong | Strong |

TABLE IV
CORRELATION THETA X APPRAISER 2

| Problem | Legend | competences | | | | |
|---|---|---|---|---|---|---|
| | | data_analysis | data_representation | problem_decomposition | abstraction | simulation |
| 1 | tau | 0.285134663 | 0.546448156 | 0.636431607 | 0.261732664 | 0.443662308 |
| | p | 0.000895726 | 4.77E-11 | 1.73E-13 | 0.001669373 | 6.96E-07 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| | Intensity | Weak | Strong | Strong | Weak | Average |
| 2 | tau | 0.505636416 | 0.385896066 | 0.418001088 | 0.510838601 | 0.11361223 |
| | p | 9.88E-16 | 2.15E-09 | 1.18E-10 | 1.30E-15 | 0.077553964 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Not rejected |
| | Intensity | Strong | Average | Average | Strong | Weak |
| 3 | tau | 0.660737588 | 0.616863733 | 0.458963521 | 0.398467672 | 0.281782669 |
| | p | 1.58E-26 | 2.05E-24 | 2.76E-13 | 3.28E-10 | 8.78E-06 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| | Intensity | Strong | Strong | Average | Average | Weak |
| 4 | tau | 0.631101432 | 0.50528655 | 0.321521716 | 0.58958336 | 0.209123951 |
| | p | 1.66E-15 | 2.69E-10 | 0.000108815 | 5.59E-13 | 0.011122595 |
| | h0 | Rejected | Rejected | Rejected | Rejected | Rejected |
| | Intensity | Strong | Strong | Average | Strong | Weak |
| 5 | tau | 0.697724662 | 0.520942718 | 0.350659116 | 0.484820009 | 0.261096146 |
| | p | 5.87E-20 | 1.30E-11 | 1.73E-05 | 1.53E-09 | 0.001340205 |
| | ho | Rejected | Rejected | Rejected | Rejected | Rejected |
| | Intensity | Strong | Strong | Average | Average | Weak |
| 6 | tau | 0.695236076 | 0.781885793 | 0.637084129 | 0.704086419 | 0.294619371 |
| | p | 2.10E-08 | 2.58E-10 | 1.59E-06 | 2.01E-08 | 0.030928921 |
| | ho | Rejected | Rejected | Rejected | Rejected | Rejected |
| | Intensity | Strong | Strong | Strong | Strong | Weak |

There was evidence that the data generated by the model and the evaluator's scores have a medium to strong correlation in the Data Analysis skills, Data Representation, and Abstraction. Although evidence has been found to stop reject the null hypothesis in most competences (except one), the correlations of the Problem decomposition with evaluator 1, and Simulation with evaluator 2 showed a weak intensity in some cases.

To understand the low intensity of the correlations, a punctual analysis was carried out regarding the behavior of the association of the Decomposition and Simulation competences.

Analyzing manually the source codes that generated the discrepant results in the comparison, it was observed that for the Problem Decomposition competence a frequency of library and header declarations (includes) was not adequate for analysis.

One of characteristics that was observed in many codes was to have several libraries declared but not used, it is attributed to the fact that it is a common practice for programmers to initialize their code with a pattern, which may contain usual libraries and predefined functions, even if not functional for

the solution of the proposed problem. The human appraiser is able to make this distinction with ease.

As they are designed, the routines created do not distinguish between libraries that have resources that are used in the source code of those that have no resources used. Thus, the model was adjusted to not account for the use of libraries in the frequency of the Problem Decomposition competency.

A similar situation occurs for the Simulation competency, also generating a difference between the evaluators and the model. There was an expectative that the comments present in the source code were part of useful information, such as documentation or output for variable state control and debugging, as in fact occurred in most cases. It is understood output for variable state control, code snippets that inform the user of the information stored in a variable in a given situation, is a debug feature used by some developers, once this information is no longer needed, this code snippet can/must be deleted or commented on.

As the routines of the present work are elaborated, it is not possible to differentiate the snippets of commented code between those that actually meet to the expected behavior of those who have another purpose. The human evaluator succeeds in making this differentiation, which in cases of unexpected use of the resource, results in a more appropriate assessment of competence. Even without adequate differentiation of use, results obtained by the model were considered satisfactory, and the use of comments for analyzing the Simulation Competence have been maintained.

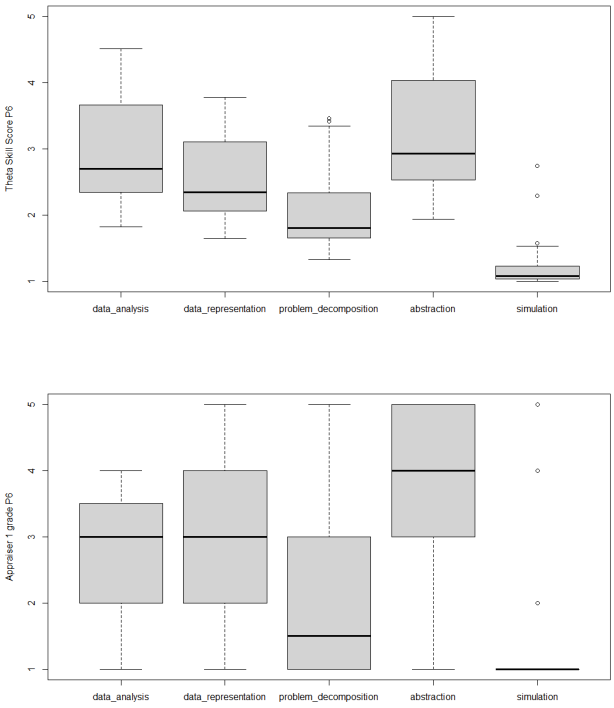
It was noticed during the experiments a significant difference between the relevance used by the model, from the TF-IDF, and the criterion of relevance of the evaluators. In the same way as when comparing the ratings of the evaluators for the competence Data Collection, in which there was a difference in the criteria used by the evaluators to assign the grade - which resulted in two very different grades even without variation in the number of tokens (mostly with only one token) - the way the model establishes relevance is different from that used by human evaluators.

For the evaluators, the relevance is restricted to the universe of a submission, while that for the model, relevance considers the context of all submissions and is influenced by the tokens not only of the analyzed competence, but also by the others present in the set of solutions. Still, for the evaluators, the quality used for the elaboration of the solution seems to be directly linked to the number of tokens for the competence under analysis in a submission.

In this context, it is easier for the evaluators to compare different submissions as to a specific competence, but by using only the number of tokens, the reference to relevance is lost. The practical result is that the humans evaluators analyze competences separately, while the model does this analysis together, in this way the skills calculated by the model are often lower than those assigned by human evaluators.

Observing the amplitude of the marks attributed by the evaluators in relation to the score of the model, it is understood that the model provides more detailed information on the competences involved in the elaboration of the source codes. The figure 2 shows the competences of problem 6 calculated by the model and by the evaluator 1. The methodology used to collect the evaluators scores (integers from 1 to 5) makes it impossible to have a scale as detailed as that calculated by the model, even so, differences in the amplitudes of the evaluations are evident. It can be seen that the variation in the score generated by the model is smoother in relation to the evaluator's grades. It is understood that this smoothing is related to relevance, for the model even if a competency performs optimally, if it is not relevant in the general context, it will not receive a high score.

Fig. 2. Box-plot Model(a) and Evaluator 1(b) Problem 6



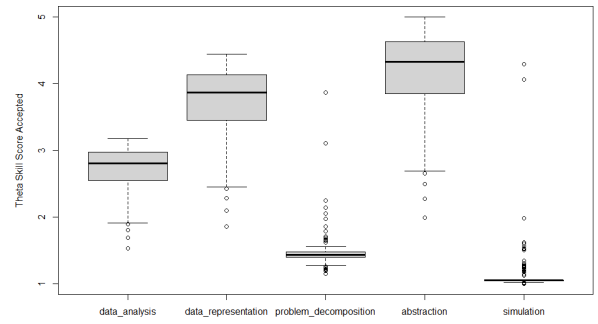

Using a simple rounding for the model's score, and comparing it to the results of the evaluators, it was identified that the incidence of equal values between the model and the evaluators is higher precisely in the skills that obtained the lowest correlations. The two competences, Problem Decomposition and Simulation, have two characteristics that were considered important for analysis: the first is that these competences are those with the lowest frequency of terms in the vast majority of submissions; the second is that they have the clearest concepts for defining competence, that is, they are the easiest to identify in a source code.

These two characteristics can be used to explain both the high number (in relation to other skills) of equal results (regarding rounded Theta scores and appraiser's grades) and the low correlations. For the original calculated Theta there is usually a variation of values, as it considers the relevance

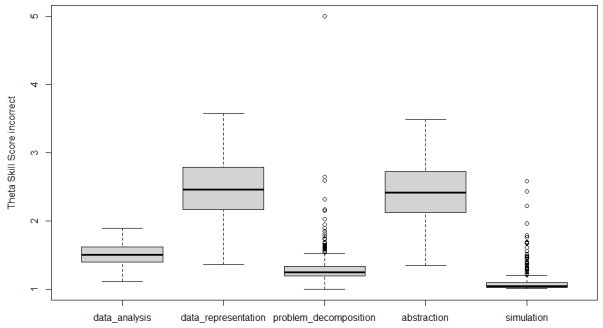
for the set of submissions as a whole, while for the human evaluation, in the absence of evidence that indicates greater competence in the elaboration of the solution, the value is usually even, fixed.

There are other conclusions that can be obtained by comparing Accepted solutions and those considered incorrect. Fig. 3 presents a graphical comparison of the set of solutions accepted as correct (a) and the set of rejected solutions (b) for problem 2. It can be seen that correct and rejected solutions share a tendency to use competences for the elaboration of solutions, however it is noteworthy that solutions considered correct have a greater intensity for some of these competences, such as problem 1, in which the skills score Data Representation and Abstraction are significantly higher in accepted solutions compared to rejected solutions.

Fig. 3. Box-plot Accepted x rejected P1



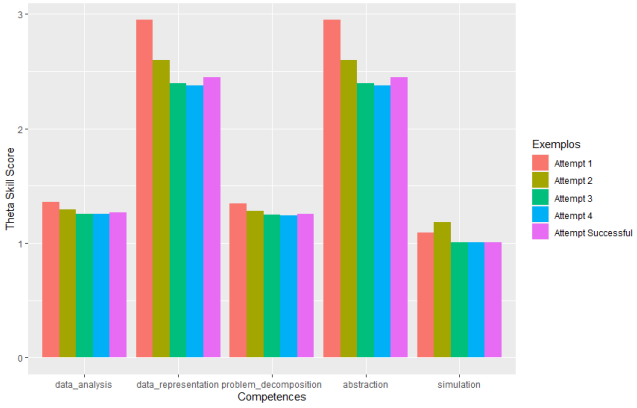

[a]




[b]

The mastery of a competency is not the primary cause of the error, or even the correctness, in programming the detail is fundamental to the point that a simple character such as a semicolon determines the success or failure of the solution. Fig. 4 shows a sequence of submissions by the same user for problem one, it is noted that as the user submits new codes his score has a slight drop, this is due to the need to add elements to his solution, making it less efficient in the face of better performing solutions, yet the figure shows that the line between a successful and an incorrect code is thin.

Fig. 4. Related submission sequence




The experiment presented has showed that the proposed methodology can be further enhanced, though, as it is, can be used as a assessing tool for competences present in source codes. Next, it is presented the final considerations and conclusions.

## V. Final Considerations

Assessing of computer programming source codes has been widely discussed over many aspects and applications, among them the educational purpose. While dynamic assessment may provide a practical way to evaluate correctness, in education it is often more rewarding to look beyond, the meaning of assessing is to provide information that can be used to foment growth and development. The present work has presented an experiment in which a model that aims to emerge the relevance of a set a competences in source codes was applied.

In general, it is understood that the model generates values differently from human evaluators, with a tendency to be smaller. Through the analysis of the obtained data, it is considered that this is an advantage in relation to the human evaluation, because the data point to indications that the model can infer a more consistent relevance, which considers the data set as a whole and is not subject to variation in subjective interpretations.

It is also understood that the model can be used to identify competences that are most influential in the success of a solution, as well as indicating the level of competences used in the development of source codes, which can be used as an assessment and monitoring tool in development of students.

Thus, the results showed that by enabling the comparison of different skills and competences present on each source code, the proposed model offers an asset for teachers to identify potential weaknesses on a student´s set of computer programming skills, and therefore be able to work on solutions for their educational development.

Future work will contemplate the refining and further development of the model and methodology. It is also aimed to incorporate the model into a educational recommendation system, in order to observe the model's behavior as an educational resource in an applied case study.

## ACKNOWLEDGEMENT

Universidade Federal do Rio Grande (FURG) and Instituto Federal de Educação Ciência e Tecnologia Sul-rio-grandense (IFSUL).

## REFERENCES

[1] P. H. Bermejo, A. O. Tonelli, R. D. Galliers, T. Oliveira, and A. L. d. S. Zambalde, "Conceptualizing organizational innovation: The case of the brazilian software industry," *Information & Management*, vol. 53, no. 4, pp. 493–503, 2016.

[2] X. Su, J. Qiu, T. Wang, and L. Zhao, "Optimization and improvements of a moodle-based online learning system for c programming," in *2016 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2016, pp. 1–8.

[3] J. Raigoza, "A study of students' progress through introductory computer science programming courses," in *2017 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2017, pp. 1–7.

[4] A. Robins, "Learning edge momentum: A new account of outcomes in cs1," *Computer Science Education*, vol. 20, no. 1, pp. 37–71, 2010.

[5] C. Cabo, "Student progress in learning computer programming: Insights from association analysis," in *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2019, pp. 1–8.

[6] A. Gomes, J. Henriques, and A. Mendes, "Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores," *Educação, Formação & Tecnologias-ISSN 1646-933X*, vol. 1, no. 1, pp. 93–103, 2008.

[7] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer science education*, vol. 15, no. 2, pp. 83–102, 2005.

[8] D. Souza, k. FELIZARDO, and E. BARBOSA, "A systematic literature review of assessment tools for programming assingnments," in *2016 IEEE 29th international Conference on Software Engeneering Education and Training (CSEET)*. IEEE, 2016, pp. 147–156.

[9] J. L. Bez, N. A. Tonin, and P. R. Rodegheri, "Uri online judge academic: A tool for algorithms and programming classes," in *2014 9th International Conference on Computer Science & Education*. IEEE, 2014, pp. 149–152.

[10] M. Mirzayanov, "Codeforces," *Statistics*, 2015. [Online]. Available: www.codeforces.com

[11] Z. Ullah, A. Lajis, M. Jamjoom, A. Altalhi, A. Al-Ghamdi, and F. Saleem, "The effect of automatic assessment on novice programming: Strengths and limitations of existing systems," *Computer Applications in Engineering Education*, vol. 26, no. 6, pp. 2328–2341, 2018.

[12] V. M. Kenski, "Avaliação da aprendizagem," *VEIGA, Ilma Passos de Alencastro. (Coord). Repensando a didática*, pp. 131–144, 1996.

[13] C. C. Luckesi, "Avaliação educacional escolar; para além do autoritarismo," *Revista da Ande*, no. 10, pp. 47–51, 1986.

[14] P. Perrenoud, "Construir competências é virar as costas aos saberes," *Revista Pátio, Porto Alegre: ARTMED, ano*, vol. 3, pp. 15–19, 1999.

[15] D. Azcona, P. Arora, I.-H. Hsiao, and A. Smeaton, "user2code2vec: Embeddings for profiling students based on distributional representations of source code," in *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*, 2019, pp. 86–95.

[16] F. Ullah, J. Wang, M. Farhan, S. Jabbar, Z. Wu, and S. Khalid, "Plagiarism detection in students' programming assignments based on semantics: multimedia e-learning based smart assessment methodology," *Multimedia Tools and Applications*, pp. 1–18, 2018.

[17] D. Ganguly, G. J. Jones, A. Ramírez-De-La-Cruz, G. Ramírez-De-La-Rosa, and E. Villatoro-Tello, "Retrieving and classifying instances of source code plagiarism," *Information Retrieval Journal*, vol. 21, no. 1, pp. 1–23, 2018.

[18] B. R. Boyce, D. H. Kraft, C. T. Meadow, D. H. Kraft, B. R. Boyce, and C. T. Meadow, *Text information retrieval systems*. Elsevier, 2017.

[19] S. E. Robertson and K. Spärck Jones, "Simple, proven approaches to text retrieval," University of Cambridge, Computer Laboratory, Tech. Rep., 1994.

[20] T. Saracevic, "Relevance: A review of and a framework for the thinking on the notion in information science," *Journal of the American Society for information science*, vol. 26, no. 6, pp. 321–343, 1975.

[21] ——, "Relevance: A review of the literature and a framework for thinking on the notion in information science. part ii," *Advances in librarianship*, vol. 30, p. 03, 2006.

[22] ——, "Relevance: A review of the literature and a framework for thinking on the notion in information science. part iii: Behavior and effects of relevance," *Journal of the American Society for information Science and Technology*, vol. 58, no. 13, pp. 2126–2144, 2007.

[23] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

[24] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[25] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[26] V. Barr and C. Stephenson, "Bringing computational thinking to k-12: what is involved and what is the role of the computer science education community?" *Acm Inroads*, vol. 2, no. 1, pp. 48–54, 2011.