A Large-scale Evaluation of a Rubric for the Automatic Assessment of Algorithms and Programming Concepts

Nathalia da Cruz Alves
Federal University of Santa Catarina
Florianópolis/Brazil
nathalia.alves@posgrad.ufsc.br

Christiane Gresse von Wangenheim
Federal University of Santa Catarina
Florianópolis/Brazil
c.wangenheim@ufsc.br

Jean Carlo Rossa Hauck
Federal University of Santa Catarina
Florianópolis/Brazil
jean.hauck@ufsc.br

Adriano Ferreti Borgatto
Federal University of Santa Catarina
Florianópolis/Brazil
adriano.borgatto@ufsc.br

ABSTRACT

As computing education makes its way into schools, there is still little research on how to assess the learning of algorithms and programming concepts as a central topic. Furthermore, in order to ensure valid instructional feedback, an important concern is the reliability and construct validity of an assessment model. Therefore, this work presents a large-scale evaluation of the CodeMaster rubric for the performance-based assessment of algorithms and programming concepts by analyzing software artifacts created by students as part of complex, open-ended learning activities. The assessment is automated through a web-based tool that performs a static analysis of the source code of App Inventor projects. Based on 88,812 projects from the App Inventor Gallery, we statistically analyzed the reliability and construct validity of the rubric. Results indicate that the rubric can be regarded as reliable (Cronbach's alpha α=0.84). With respect to construct validity, there also exists an indication of convergent validity based on the results of a correlation and factor analysis. This indicates that the rubric can be used for a valid assessment of algorithm and programming concepts of App Inventor programs as part of a comprehensive assessment completed by other assessment methods. The results can guide the improvement of assessment models, as well as support the decision on the application of the rubric in order to support computing education in K-12.

CCS CONCEPTS

• Social and professional topics~Computational thinking • Social and professional topics~Student assessment • Social and professional topics~K-12 education

SIGCSE'20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03 $15.00
https://doi.org/10.1145/3328778.3366840

KEYWORDS

Automated grading; Algorithms and programming; Performance-based assessment; Rubric

ACM Reference format:

Nathalia da C. Alves, Christiane G. von Wangenheim, Jean C. R. Hauck, and Adriano F. Borgatto. 2020. A Large-scale Evaluation of a Rubric for the Automatic Assessment of Algorithms and Programming Concepts. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE'20), March 11-14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366840

1. INTRODUCTION

Computational Thinking (CT) is a competence that refers to the thought processes involved in creating algorithmic solutions that can be performed by a computer [1]. It is regarded as a key competence that students need to develop in order to succeed in a rapidly changing, digital society [1]. According to the CSTA Computer Science K-12 Framework [2], CT practices can be applied to many areas, however, computer science offers unique opportunities to develop CT. There is a strong relationship between CT practices and algorithms and programming concepts (Figure 1) that allows exploring this relationship by using the concepts present in any programming language [2].

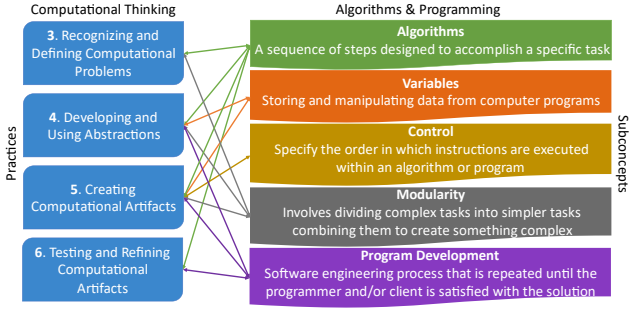

Figure 1: Relationship between CT practices and algorithms and programming concepts [2]

All these algorithms and programming concepts, also known as algorithm and thinking skills, are strongly related to CT practices from the Computer Science K-12 Framework [2].

Teaching CT and algorithmic thinking has been a focus of worldwide efforts of computing education in K-12 [3][4]. Many of those initiatives focus on teaching programming that is not solely a basic part of computing, but also a key tool for supporting cognitive tasks involved in CT [3]. In order to introduce programming in K-12, typically visual block-based programming environments, such as Scratch, BYOB/Snap! or App Inventor are used [5]. Diverse instructional strategies are applied, yet, often in a constructivist context, a problem-based learning strategy is adopted posing programming activities as open-ended ill-structured problems [6]. These activities aim to stimulate the development of higher-order thinking competencies not prescribing a correct or best solution in advance and giving students more freedom to choose abstract concepts for creating a solution [7]. Examples are activities in which students create their own games or mobile applications to solve real-world problems [8].

Important for any learning process, assessment and grading can guide the students' learning providing feedback to both the student and the teacher [9][10][11][12]. Yet, despite the many efforts to address CT assessment [3][13][14], there is currently no consensus on strategies to assess CT concepts [13][14][15][16]. Due to the abstract nature of the construct being measured, the assessment of CT is particularly complex [17]. Different approaches and frameworks have been proposed [14], yet, for assessing complex, ill-structured activities as part of problem-based learning, authentic assessments seem a more appropriate means than traditional assessments [18][19].

Authentic assessment, specifically performance-based assessment, measures competencies based on artifacts created by students as a result of open-ended tasks, such as programming a software artifact [20]. By defining outcome-oriented assessment indicators it allows to assess the product of the students' performance with respect to the learning objectives that specify what the student should learn, understand or be able to do as a result of an instructional unit [21]. In the context of the assessment of CT, such outcome-oriented indicators assume that measurable attributes can be extracted from the software artifact created by the student. Performance levels for the achievement of learning outcomes defining assessment criteria and indicators are typically defined as rubrics [22]. Rubrics, used for assessing programming activities, map a score to the ability to develop a software artifact indirectly inferring the achievement of CT competencies [15][23][24]. These scores can also be converted into grades. Examples of rubrics that assess some CT aspects include Dr. Scratch [25], Ninja Code Village [26], Grover et al. rubric [27], Basu rubric [28] and the mobile CT rubric [24].

In order to support their application in practice, some of these rubrics have been automatized, such as Dr. Scratch [25] and Ninja Code Village [26]. These tools measure CT by performing a static code analysis by counting the kind and the number of command blocks used with respect to algorithms and programming concepts, such as logic, abstraction, control flow, etc., and thus, quantifying CT practices. Yet, most of these rubrics and tools for assessing CT focus on Scratch [14].

Research on the assessment of code created with other popular block-based programming environments including App Inventor is still scarce. Sherman and Martin propose a mobile CT rubric aiming at the assessment of mobile CT based on the analysis of App Inventor programs [24][29]. This rubric has been designed to assess growth in sophistication of mobile CT patterns in App Inventor programs including 6 criteria on CT in general and 8 specific criteria on mobile CT considering that mobile platforms provide situatedness of computing. An evaluation of the rubric based on the analysis of 45 apps from 18 students in higher education illustrates the efficacy of the rubric to demonstrate varying degrees of mobile CT skills [24]. However, the rubric has been developed for App Inventor Classic, retired in 2015 and being replaced by App Inventor 2 including new features (such as maps, media, etc.) not covered by that rubric. Grover et al. [27] propose a rubric to assess Scratch and App Inventor projects with respect to 5 main dimensions: general factors, mechanics of design, user experience, basic coding & constructs, and advanced coding constructs. They used the rubric to assess students' projects from 6th to 8th grade and drawn conclusions on the sequencing of some concepts in K-12 computer science trends, as well as a comparison between Scratch and App Inventor. Another multidimensional rubric for analyzing open-ended block-based programming projects that integrates the assessment of front-end project design and back-end sophistication of the use of coding constructs has been presented by Basu [28]. The rubric was also used to assess students' projects from 6th to 8th grade in order to give computer science educators an overview of what students need to have more support and what can be done to leverage programming environments to provide this support. Yet, none of these rubrics were automated and analyzed regarding their reliability and validity.

Thus, based on these existing CT rubrics and taking into consideration new features of App Inventor, we propose an automated performance-based assessment rubric and grader CodeMaster. It enables an analysis of the code of App Inventor programs supported by a free web-based tool providing feedback to students and teachers in the form of a CT score on programming projects. The model has been developed based on a systematic mapping study [14] following an instructional design process [30] and the procedure for rubric definition proposed by Goodrich [31]. A preliminary evaluation conducting a user test with K-12 teachers and students indicated that it can be a useful, usable and efficient tool to support the assessment of App Inventor projects [32].

Yet, a question that remains is if the CodeMaster rubric allows assessing CT in a reliable and valid manner. Therefore, we conduct an extensive evaluation of the CodeMaster rubric with respect to its reliability and construct validity, being

fundamental issues with respect to measurement instruments such as rubrics [33].

2. CODEMASTER RUBRIC AND TOOL

CodeMaster is a model for the assessment of algorithms and programming concepts related to CT practices and mobile concepts. It is designed to be applicable in the context of complex and ill-structured programming tasks without a single correct solution, e.g., students developing their own apps to solve a community problem. CodeMaster adopts an authentic assessment strategy. It measures the students' performance based on the created software artifacts as outcomes of programming tasks. It is based on a rubric that measures indicators of the learning outcome in order to evaluate whether the software artifact produced by students demonstrates that they have learned algorithms and programming concepts (Table 1). The items were decomposed based on learning objectives from levels 1B to 3A from the CSTA Computer Science K-12 Framework, which are designed to assess students' projects from grades 3-10 [2].

Differences with respect to the Mobile CT rubric [24][29] are due to new features in App Inventor 2 (such as maps and other components) that were inexistent when the Mobile CT rubric was created. No criterion on parallelism is included, as it is not applicable to App Inventor programs [34]. Focusing exclusively on assessment based on the created software artifacts, criteria related to the development process and CT perspectives, as proposed by Brennan and Resnick [15], are also not considered. Aspects related to subjective criteria are also not covered by the rubric, as it is difficult to automatize such project characteristics that are typically assessed by a human, such as novelty [28].

For each performance level, observable behaviors are described on an ordinal scale, ranging from "criterion is not (or minimally) present" to an advanced usage of the criterion based on the sources appointed in Table 1. As a result, a score is assigned for each criterion, referred to as item. A total score is calculated through the sum of the partial scores ranging from [0, 40].

The assessment and grading of App Inventor projects based on the model are automated through a free web-based system available on http://apps.computacaonaescola.ufsc.br:8080/. Students can use the tool throughout the learning process in

Table 1: CodeMaster rubric for assessing algorithms and programming based on the analysis of App Inventor projects

CT Sub-dimension	Criterion	Performance Level				Source Reference
		0	1	2	3	
Algorithms and Programming concepts	**1. Operators**	No operator blocks are used.	Arithmetic operator blocks are used.	Relational operator blocks are used.	Boolean operator blocks are used.	[15][27]
	2. Variables	No use of variables.	Modification or use of predefined variables.	Creation and operation with variables.	-	[2]:1B-AP-09
	3. Strings	No use of strings.	Use of creating string block to change design elements texts.	Creation and operation with strings.	-	[2]:1B-AP-09
	4. Naming	Few or no names are changed from their defaults.	10 to 25% of the names are changed from their defaults.	26 to 75% of the names are changed from their defaults.	More than 75% of the names are changed from their defaults.	[2]:2-AP-11
	5. Lists	No lists are used.	One single-dimensional list is used.	More than one single-dimensional list is used.	Lists of tuples are used.	[2]:3A-AP-14
	6. Data persistence	Data are stored only in variables or UI component properties, and do not persist when app is closed.	Data is stored in files.	Local database is used.	Web database is used.	[24][29]
	7. Events	No type of event handlers is used.	One type of event handlers is used.	Two or three types of event handlers are used.	More than three types of event handlers are used.	[2: 1B-AP-10
	8. Loops	No use of loops.	Simple loops are used.	'For each' loops with simple variables are used.	'For each' loops with list items are used.	[2]:1B-AP-10 [2]:2-AP-12
	9. Conditional	No use of conditionals.	Uses 'if' structure.	Uses one 'if then else' structure.	Uses more than one 'if then else' structure.	[2]:1B-AP-10 [2]:2-AP-12
	10. Synchronization	No use of timer for synchronization.	Use of timer for synchronization.	-	-	[15]
	11. Procedural Abstraction	No use of procedures.	One procedure is defined and called.	More than one procedure defined.	There are procedures for code organization and re-use.	[2]:1B-AP-11 [2]:2-AP-13
Mobile concepts	**12. Sensors**	No use of sensors.	One type of sensor is used.	Two types of sensors are used.	More than two types of sensors are used.	[24][29]
	13. Drawing and Animation	No use of drawing and animation components.	Uses canvas component.	Uses ball component.	Uses image sprite component.	[24][29]
	14. Maps	No use of maps.	Use of a map block	Use of map markers blocks.	-	[24][29]
	15. Screens	Single screen with visual components, whose state is not changed programmatically.	Single screen with visual components, whose state is changed programmatically.	Three screens with visual components of which at least one is programmed to change state.	Four screens with visual components of which at least two are programmed to change state.	[24][29]

order to obtain immediate feedback on their projects (Figure 2). In addition, teachers can use the tool in order to assess and grade projects of an entire class as part of a comprehensive assessment as suggested by Brennan and Resnick [15] and Grover and Pea [35].

Figure 2: CodeMaster grade for an App Inventor project

3. RESEARCH METHOD

Following the Goal Question Metric (GQM) approach [36], the objective of this study is to analyze the CodeMaster rubric in order to evaluate its reliability and construct validity for the assessment of CT competencies related to algorithms and programming concepts from the researchers' perspective in the context of K-12 computing education. Based on this objective, the following analysis questions are derived:

Reliability

AQ1: Is there evidence of internal consistency of the CodeMaster rubric?

Construct Validity

AQ2: Is there evidence of convergent validity of the CodeMaster rubric?

AQ3: How do underlying factors influence the responses on the items of the CodeMaster rubric?

Data collection. In order to optimize the sample size, we downloaded the publicly available and accessible apps from the App Inventor Gallery in June 2018. As a result, we obtained the source-code from 88,864 App Inventor apps. We analyzed these projects using the CodeMaster tool. Out of the 88,864 downloaded projects, 88,812 were successfully analyzed. 52 projects failed to be analyzed due to technical difficulties. The collected data were pooled in a single sample in order to validate the CodeMaster rubric (rather than a specific app).

Data analysis. In order to evaluate the reliability and construct validity of the CodeMaster rubric, we used different statistical methods. As reliability refers to the degree of consistency or stability of the instrument criteria on the same quality factor, we estimate internal consistency calculating Cronbach's alpha coefficient [37] in order to evaluate the consistency of results across criteria within a measurement instrument. Construct validity is defined as the instrument's ability to actually measure what it purports to measure, involving convergent validity obtained through the degree of correlation between the instrument criteria [33]. We also performed a factor analysis in order to determinate how many factors underlie the set of items of the CodeMaster rubric, conducting the analysis proposed by Brown [38]. This allows identifying the amount to which each item is correlated with each subdimension through the factor loading.

4. ANALYSIS

4.1 Is there evidence of internal consistency of the CodeMaster rubric?

We analyzed reliability by measuring the internal consistency of the CodeMaster rubric through Cronbach's alpha coefficient [37]. Cronbach's alpha coefficient measures indirectly to what extent a set of items measures a single quality factor. Thus, in our case, whether the CodeMaster rubric measures the same factor: the assessment of CT competencies related to algorithms and programming concepts. Typically, values ranging from 0.70 to 0.95 are considered acceptable indicating internal consistency (values between $0.7 < \alpha \leq 0.8$ are acceptable, $0.8 < \alpha \leq 0.9$ are good, and $\alpha \geq 0.9$ are excellent) [39]. Analyzing the 15 items of the CodeMaster rubric using the 88,812 App Inventor projects, we obtained a satisfactory value of Cronbach's alpha (α=0.84).

4.1 Is there evidence of convergent validity of the CodeMaster rubric?

In order to demonstrate convergent validity, we expect the items of the rubric to have a medium or high correlation with all other items [39]. We analyzed this question using the method of corrected item-total correlation that compares one item to every other one of the rubric. Following Cohen [40], a correlation is considered satisfactory if the correlation coefficient is greater than 0.29. We also analyze the value of Cronbach's alpha if an item was deleted, expecting no substantial increase [39]. The values of the item-total correlation and the values of Cronbach's alpha [37] deleting the respective item are presented in Table 2.

Table 2. Results of the analysis of the item-total correlation with 88,812 App Inventor projects

Criterion (or Item)	Item-total correlation	Cronbach alpha
1. Operators	0.694	0.82
2. Variables	0.686	0.82
3. Strings	0.583	0.83
4. Naming	0.585	0.82
5. Lists	0.364	0.84
6. Data persistence	0.325	0.84
7. Events	0.596	0.82
8. Loops	0.286	0.84
9. Conditional	0.618	0.82
10. Synchronization	0.562	0.83
11. Procedural Abstraction	0.548	0.83
12. Sensors	0.448	0.84
13. Drawing and Animation	0.376	0.84
14. Maps	**0.015**	**0.85**
15. Screens	0.324	0.84

Most values of the item-total correlations are above 0.29, demonstrating an acceptable internal consistency. The degree of correlation between items indicates that the items measure the same dimension, thus, demonstrating convergent and discriminant validity.

Only item 14 (Maps) demonstrates a low correlation, which can be explained by the fact that maps are a resource that has been recently added to App Inventor in 2018, and, thus, may be underrepresented in our dataset. Item 8 (Loops) shows a correlation of 0.28, very close to the expected minimum value (0.29). This may be due to the fact that loop commands are not widely used in App Inventor programs [34][41]. All other items show a decrease in Cronbach's alpha value if removed and demonstrate sufficient item-total correlation, indicating the validity of the rubric.

4.2 How do underlying factors influence the responses on the items of the CodeMaster rubric?

We performed a factor analysis in order to identify the number of underlying factors that influence the items of the CodeMaster rubric. We assume that the rubric is influenced by two factors: algorithms and programming concepts and mobile concepts, both related to CT.

In order to check the possibility to perform a factor analysis, we used the Kaiser-Meyer-Olkin (KMO) index [38]. It measures the sampling adequacy with values between 0 and 1. A value near 1.0 supports a factor analysis and anything less than 0.5 is not likely suitable for useful factor analysis [38]. Analyzing the items of the CodeMaster rubric, we obtained a KMO index of 0.83, demonstrating that factor analysis is suitable in this case.

The number of factors retained is decided by applying factor analysis [42]. We used parallel analysis, a method for determining the number of components or factors to retain in which factors with eigenvalues greater than 1 may be significant. Conducting the parallel analysis, results show that there is one preponderant factor, however, by showing 3 eigenvalues above the red line, the scree plot suggests that there may be 3 underlying factors (Figure 3).

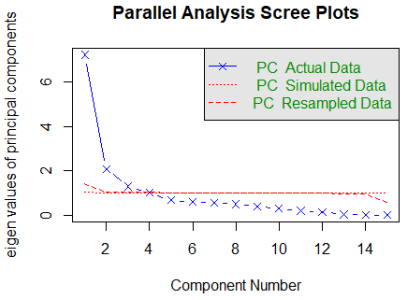

Figure 3: Parallel analysis scree plots of the CodeMaster rubric

Thus, in order to decide which items are loaded in each factor, we use the Oblimin rotation method, in which the factors are allowed to be correlated [43]. Table 3 shows the factor loadings of the items for each of the 3 retained factors. Based on Comrey and Lee [44], we use more stringent cut-offs from 0.32 (poor), 0.45 (acceptable), 0.55 (good), 0.63 (very good), or 0.71 (excellent). Thus, good factor loadings (above 0.55) are marked in bold in Table 3.

Table 3. Factor loadings for 3 factors

Criterion (or item)	Factor 1	Factor 2	Factor 3
1. Operators	0,325	0,074	**0,795**
2. Variables	0,453	0,337	**0,763**
3. Strings	-0,028	-0,100	**0,801**
4. Naming	0,198	0,174	**0,659**
5. Lists	-0,070	0,123	**0,690**
6. Data persistence	-0,093	-0,156	**0,786**
7. Events	0,178	-0,157	**0,868**
8. Loops	-0,004	0,209	**0,768**
9. Conditional	0,150	0,048	**0,807**
10. Synchronization	**0,710**	-0,336	**0,616**
11. Procedural Abstraction	0,406	0,241	**0,779**
12. Sensors	**0,713**	-0,432	0,453
13. Drawing and Animation	**0,752**	0,298	0,351
14. Maps	-0,123	-0,389	0,403
15. Screens	-0,158	-0,339	**0,702**

Analyzing the factor loadings (Table 3), factor 1 seems to be more related to mobile concepts from App Inventor, grouping item 12 (Sensors) and item 13 (Drawing and Animation) in this factor. In addition, item 10 (Synchronization) also presents a high factor loading in this first factor. However, it also has a high factor loading in the third factor.

Factor 2 presents poor factor loadings for all items. No item demonstrates a factor loading above 0.45, indicating that there is no such factor.

Factor 3 is more related to algorithms and programming concepts related to CT practices as defined by the CSTA [2]. Most of the items of the CodeMaster rubric have a high factor loading in this third factor. Item 14 (Maps), semantically more related to the first factor, presents its largest factor loading in the third factor. However, as this item is related to a resource added to App Inventor in 2018, it may be underrepresented in our dataset in which only 0,08% projects (72 out of 88,812 projects) use this feature, this may have generated noise in the calculation. As a result, the factor loadings confirm that the model has only two subdimensions as originally proposed in table 1 (algorithms and programming concepts and mobile concepts).

Aiming ultimately at assessing CT, it is important to determine how much the items are loading into one single factor. Table 4 shows the factor loadings of the items with respect to one single factor. Most of the items present value above the threshold of 0.55. Only item 14 (Maps) has a low factor loading, which again can be explained by the possibility of this item being underrepresented in our dataset.

Table 4. Factor loadings for one single factor

CRITERIA (OR ITEM)	FACTOR LOADING
1. Operators	**0.875**
2. Variables	**0.868**
3. Strings	**0.697**
4. Naming	**0.703**
5. Lists	**0.590**
6. Data persistence	**0.679**
7. Events	**0.860**
8. Loops	**0.721**
9. Conditional	**0.807**
10. Synchronization	**0.855**
11. Procedural Abstraction	**0.882**
12. Sensors	**0.669**
13. Drawing and Animation	**0.614**
14. Maps	0.262
15. Screens	**0.574**

5. IMPLICATIONS AND LIMITATIONS

The obtained results, in general, indicate good reliability and construct validity of the CodeMaster rubric for the assessment of CT competencies related to algorithms and programming concepts and mobile concepts. Unsatisfactory results regarding item 14 (Maps) may be due to its underrepresentation in the dataset. All other items presented good results with respect to both, reliability and validity.

In addition, the items also present consistent results related to the underlying factors of the rubric, and the factor analysis shows that the items have high factor loadings in two subdimensions, as originally proposed. However, item 15 (Screens) demonstrated a bigger factor loading on algorithms and programming concepts instead of mobile concepts. This could be due to the definition of this item, which is related to "the app state changing programmatically" as presented in Table 1. Thus, for this change happen, it is required that the app contains many algorithms and programming concepts in order to provide screen change to the user.

One item presents high factor loading in both factors. Item 10 (Synchronization) seems to be related to algorithms and programming concepts as defined by the CSTA [2] as well as to mobile concepts from App Inventor. As a result, we can conclude an assessment model composed of two sub-dimensions as shown in Figure 4.

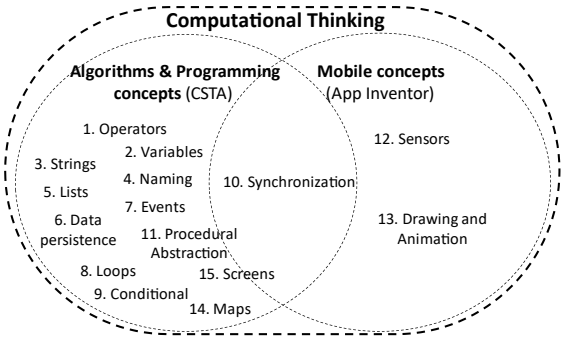

Figure 4: CodeMaster rubric items grouped based on the results of the factor analysis

In general, the results of the analysis show that the CodeMaster rubric represents a reliable instrument and, with the exception of the Maps item, also demonstrates a good correlation. This indicates that the rubric can be used for a valid assessment of algorithms and programming concepts based on a code analysis of App Inventor programs. Yet, it is important to point out that the rubric represents only one alternative for measuring these dimensions of CT, and a more comprehensive assessment should be completed by other assessment methods, such as interviews, peer reviews, presentations, etc. as suggested by Brennan and Resnick [15] and Grover and Pea [35].

Threats to validity. In order to minimize the impact on our research, we identified potential threats and applied mitigation strategies. In order to mitigate threats related to the study design, we adopted a systematic methodology following the GQM approach [36]. Another issue refers to the quality of data pooled into a single sample considering the standardization of the data. As our study is limited to assessments using the CodeMaster tool, this risk is minimized as all analyses were performed in an automated way using the same rubric. Another risk is the pooling of data from various contexts. The apps of the dataset come from diverse contexts from the App Inventor community worldwide and no further information on the background of the creators is available. However, as the objective is to analyze the validity of the rubric in a context-independent way, this is not considered an issue here. Another threat to external validity is associated with the sample size and diversity of the data used. Our analysis is based on projects collected from the App Inventor Gallery, involving a sample of 88,812 apps from the App Inventor community worldwide. This is considered a satisfactory sample size that allows the generation of significant results. Another issue refers to which the extent the data and analysis are influenced by the researchers. In order to mitigate this threat, we adopted a systematic methodology, defining clearly the study objective, the data collection, and statistical analysis. We also carefully selected the statistical methods following the approach proposed by DeVellis [39] for the construction of measurement scales in alignment with procedures for the analysis of internal consistency and construct validity of measurement instruments [45].

6. CONCLUSION

In general, the results of this large-scale evaluation show that the CodeMaster rubric represents an instrument with acceptable reliability and validity that can be used for the assessment of algorithms and programming concepts of App Inventor applications as part of computing education in schools. Supported through the online tool CodeMaster, it further provides automated support that helps to ensure consistency and accuracy of assessment results as well as to eliminate bias. Furthermore, it can also reduce the teachers' workload and leaving them free to spend more time on other activities with students as well as to conduct complementary assessments on factors that are not easily automated, such as creativity.

ACKNOWLEDGMENTS

We would like to thank all researchers from the MIT App Inventor team, who provided support for the access to the App Inventor Gallery. The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq) - Grant No.: 302149/2016-3.

REFERENCES

[1] J. Wing. 2006. Computational Thinking. Communications of the ACM. 49(3), 33-36.

[2] Computer Science Teachers Association. 2016. K-12 Computer Science Framework. Retrieved August 09, 2019 from https://k12cs.org/

[3] S. Grover, and R. Pea. 2013. Computational Thinking in K–12: A review of the state of the field. Educational Researcher, 42(1), 38-43.

[4] Y. Kafai, and Q. Burke. 2013. Computer programming goes back to school. Phi Deltan Kappan, 95(1), 61–65.

[5] S. Y. Lye and J. H. L Koh. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12?. Computers in Human Behavior, 41(C), 51–61.

[6] D. Reed. 2002. The use of ill-defined problems for developing problem-solving and empirical skills in CS1. Journal of Computing Sciences in Colleges, 18(1), 121-133.

[7] D. Fortus, R. C. Dershimer, J. S. Krajcik, R. W. Marx, R. Mamlok-Naaman. 2004. Design-based science and student learning. Journal of Research in Science Teaching, 41(10), 1081–1110.

[8] S. B. Fee, and A. M. Holland-Minkley. 2010. Teaching Computer Science through Problems, not Solutions. Computer Science Education, 20(2), 129-144.

[9] J. Hattie, and H. Timperley. 2007. The power of feedback. Review of Educational Research, 77(1), 81-112.

[10] V. J. Shute. 2008. Focus on formative feedback. Review of Educational Research, 78(1), 153-189.

[11] P. Black, and D. Wiliam. 1998. Assessment and classroom learning. Assessment in Education: Principles, Policy & Practice, 5(1), 7–74.

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

[13] S. Grover, S. Cooper, and R. Pea. 2014. Assessing Computational Learning in K-12. In Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE '14). ACM, NY, USA, 57-62.

[14] N. da C. Alves, C. Gresse von Wangenheim, and J. C. R. Hauck. 2019. Approaches to assess computational thinking competences based on code analysis in K-12 education: A systematic mapping study. Informatics in Education, 18(1), 17-39.

[15] K. Brennan and M. Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In Proceedings of the Annual Meeting of the American Educational Research Association

[16] S. Grover. 2017. Assessing Algorithmic and Computational Thinking in K-12: Lessons from a Middle School Classroom. In: Rich P., Hodges C. (eds) Emerging Research, Practice, and Policy on Computational Thinking. Educational Communications and Technology: Issues and Innovations. Springer, Cham.

[17] A. Yadav, D. Burkhart, D. Moix, E. Snow, P. Bandaru, and L. Clayborn. 2015. Sowing the Seeds: A Landscape Study on Assessment in Secondary Computer Science Education. In Proceedings of the CSTA Annual Conference.

[18] H. Torrance. 1995. Evaluating authentic assessment: Problems and possibilities in new approaches to assessment. Buckingham: Open Uni. Press.

[19] J. D. Ward and C. L. Lee. 2002. A review of problem-based learning. Journal of Family and Consumer Sciences Education, 20(1), 16-26.

[20] G. P. Wiggins. 1993. Assessing student performance: Exploring the purpose and limits of testing. In The Jossey-Bass education series. San Francisco: Jossey-Bass.

[21] A. Driscoll and S. Wood. 2007. Developing Outcomes-Based Assessment for Learner Centered Education: a Faculty Introduction. Sterling: Stylus Publishing.

[22] R. McCauley. 2003. Rubrics as assessment guides. SIGCSE Bull. 35(4) (December 2003), 17-18.

[23] S. Srikant and V. Aggarwal. 2013. Automatic Grading of Computer Programs: A Machine Learning Approach. In Proceedings of the 2013 12th International Conference on Machine Learning and Applications - Volume 01 (ICMLA '13), Vol. 1. IEEE Computer Society, Washington, DC, USA, 85-92.

[24] M. Sherman and F. Martin. 2015. The assessment of mobile computational thinking. Journal of Computing Sciences in Colleges, 30(6), 53–59.

[25] J. Moreno-León and G. Robles. 2015. Dr. Scratch: a Web Tool to Automatically Evaluate Scratch Projects. In Proceedings of the Workshop in Primary and Secondary Computing Education (WiPSCE '15). ACM, NY, USA, 132-133.

[26] G. Ota, Y. Morimoto and H. Kato. 2016. Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 238-239.

[27] S. Grover, S. Basu, and P. Schank. 2018. What We Can Learn About Student Learning From Open-Ended Programming Projects in Middle School Computer Science. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). ACM, NY, USA, 999-1004.

[28] Satabdi Basu. 2019. Using Rubrics Integrating Design and Coding to Assess Middle School Students' Open-ended Block-based Programming Projects. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19). ACM, NY, USA, 1211-1217.

[29] M. Sherman, F. Martin, L. Baldwin, J. DeFilippo. 2014. App Inventor Project Rubric – Computational Thinking through Mobile Computing. Retrieved July 06, 2019 from https://nsfmobilect.files.wordpress.com/2014/09/mobile-ct-rubric-for-app-inventor-2014-09-01.pdf

[30] R. M. Branch. 2010. Instructional Design: The ADDIE Approach. New York: Springer.

[31] H. Goodrich. 1996. Understanding Rubrics. Educational Leadership, 54(4), 14–18.

[32] C. Gresse von Wangenheim, J. C. R. Hauck, M. F. Demetrio, R. Pelle, N. da C. Alves, H. Barbosa, L. F. Azevedo. 2018. CodeMaster – Automatic Assessment and Grading of App Inventor and Snap! Programs. Informatics in Education, 17(1), 117-150.

[33] E. G. Carmines and R. A. Zeller. 1982. Reliability and validity assessment (5th ed.). Beverly Hills: Sage Publications Inc.

[34] F. Turbak, M. Sherman, F. Martin, D. Wolber, and S. C. Pokress. 2014. Events-first programming in APP inventor. Journal of Computing Sciences in Colleges, 29(6), 81-89.

[35] S. Grover, & R. Pea. 2015. "Systems of Assessments" for Deeper Learning of Computational Thinking in K-12. In Proceedings of the Annual Meeting of the American Educational Research Association.

[36] V. R. Basili, G. Caldiera, H. D. Rombach. 1994. The Goal Question Metric Approach. In Encyclopedia of Software Engineering, Wiley.

[37] L. J. Cronbach. 1951. Coefficient alpha and the internal structure of tests. Psychometrika, 16(3), 297–334.

[38] T. A. Brown. 2006. Confirmatory factor analysis for applied research. New York: The Guilford Press.

[39] R. F. DeVellis. 2003. Scale development: theory and applications. Thousand Oaks: SAGE Publications.

[40] Cohen, J. 1998. Statistical Power Analysis for the Behavioral Sciences. New York: Routledge Academic.

[41] B. Xie and H. Abelson. 2016. Skill progression in MIT App Inventor. Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (pp. 213–217).

[42] L. W. Glorfeld. 1995. An improvement on Horn's parallel analysis methodology for selecting the correct number of factors to retain. Educational and Psychological Measurement, 55(3), 377-393.

[43] J. E. Jackson. 2014. Oblimin Rotation. In Wiley StatsRef: Statistics Reference Online (eds N. Balakrishnan et al.).

[44] A. L. Comrey and H. B. Lee. 1992. A first course in factor analysis (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

[45] W. M. Trochim and J. P. Donnelly. 2008. Research methods knowledge base (3rd ed.). Mason, OH: Atomic Dog Publishing.