Modeling Creativity in Visual Programming: From Theory to Practice

Anastasia Kovalkov
Ben-Gurion University of the Negev
Kovalkov@post.bgu.ac.il

Benjamin Paaßen
Humboldt-University of Berlin
benjamin.paassen@hu-berlin.de

Avi Segal
Ben-Gurion University of the Negev
kobig@bgu.ac.il

Kobi Gal
Ben-Gurion University of the Negev
University of Edinburgh
kobig@bgu.ac.il

Niels Pinkwart
Humboldt-University of Berlin
niels.pinkwart@hu-berlin.de

ABSTRACT

Promoting creativity is considered an important goal of education, but creativity is notoriously hard to define and measure. In this paper, we make the journey from defining a formal creativity and applying the measure in a practical domain. The measure relies on core theoretical concepts in creativity theory, namely fluency, flexibility, and originality, We adapt the creativity measure for Scratch projects. We designed a machine learning model for predicting the creativity of Scratch projects, trained and evaluated on ratings collected from expert human raters. Our results show that the automatic creativity ratings achieved by the model aligned with the rankings of the projects of the expert raters more than the experts agreed with each other. This is a first step in providing computational models for describing creativity that can be applied to educational technologies, and to scale up the benefit of creativity education in schools.

Keywords

Creativity, Creativity Tests, Visual Programming Environments

1. INTRODUCTION

Modern education generally tries to foster creativity in student problem solving [7, 11, 17]. There is wide agreement that creative solutions must not only solve the task but should additionally be *original*, i.e. distant from usual solutions to the task, *flexible*, i.e. employ very different concepts, and *fluent*, i.e. employ many concepts [10, 23]. However, creativity is notoriously hard to quantify in practice [7]. When being confronted with two student solutions for a given learning task, different teachers may well disagree which one is more creative [14].

In this paper, we make the journey from a definition of creativity which relate to prior concepts in the literature [23], to applying the definition to the Scratch programming environment, and using the measure to automatically quantifying the creativity score of projects in Scratch. We formalize originality of a product as the distance to usual solutions, flexibility as the distance between concepts in the student's solution, and fluency as the distance to an empty solution.

We apply the formalization to automatically measure the creativity on a set of projects from the popular visual programming language Scratch [13]. Using machine learning, we build a model that predicts the creativity ratings of Scratch projects using fluency, flexibility, and originality measures of our approach. We compare these automatic creativity ratings to those of five human experts, which were collected using a comprehensive user study. We find that the automatic ratings agree with the rankings of the experts more than the experts agreed with each other. We provide several examples that highlight the benefit of the model in light of the fact that human raters may disagree on the degree of creativity of Scratch projects.

The contribution of this work is in providing an automatic framework for defining and detecting creativity, that can scale up teacher's abilities to support creative thinking in students.

2. RELATED WORK

Prior works on measuring creativity have mostly been concerned with psychological tests, such as Williams' tests on creative thinking [25] or the Torrance test of creative thinking [10, 23]. However, such tests do not account for changes in creative ability, motivation, knowledge, and social context over time [2]. Accordingly, one should wish to measure creativity often and monitor the development across changing circumstances. This could be supported by automatic creativity assessment, towards which we work in this paper.

To measure creativity at one specific point in time, we follow Torrance's work and grade creativity on three scales, namely fluency, flexibility, and originality [10, 23]. Historically, these three scales grew out of Guilford's model of the structure of intellect [6], which includes divergent production, i.e. the skill of generating a wide variety of ideas on the same topic. In particular, fluency refers to the sheer amount of ideas generated, flexibility to the number of distinct classes of ideas, and originality to the infrequency of the ideas compared to a general sample of the population. In addition to these three scales, Torrance's tests also include scales regarding the abstractness of generated ideas, the elaboration on ideas, and the resistance to premature closure during the generation process [10]. In this work, we stick to fluency, flexibility, and originality because they permit quite a direct formalization in mathematical and computeable terms. We further cover elaboration, to some extent, in our notion of fluency.

Multiple works focused on using technologies to infer creative thinking in numerous educational disciplines, such as math and programming [21, 12]. Previous research has shown that creativity is related to positive learning gains, and using technology to generate creativity is an active field of study [24]. Hershkovitz et al. [8, 9] examined the relationship between creativity and computational thinking within a block-based multi-level game environment for children's programming. Their findings show that creativity can contribute to the acquiring of computational thinking and can also be transferred across domains, stressing the importance of fostering creativity while promoting computational thinking.

Finally, the application domain of our work is Scratch, a block-based visual programming language targeted primarily at children. The environment allows users to create interactive stories, games, and animations [13]. Scratch blocks are designed to fit together in ways that make syntactic sense which generates the program logic. Users can use a wide variety of pre-defined basic code blocks, such as *When Key Pressed*, *Move* etc. Furthermore, programmers can use additional blocks such as *Pen Down* and *Language* from existing extensions such as 'Pen' and 'Translate', as well as define custom blocks. The environment allows the use of external data through importing images, music recordings, captured voices, and user-specific graphics [18]. By using Scratch, which is designed to enable creative expression in terms of code, graphical and audio aspects, we can expand the identification of creativity beyond the programming aspect [3, 5, 12].

3. COMPUTING CREATIVITY IN SCRATCH

In this section, we describe how we measure creativity of code and visual aspects of Scratch programs.

A Scratch project consists of the project's background called *stage* and the objects that appear on it called *sprites*[1]. Figure 1 presents a sample project, a game called 'Scratch in Scratch'. As its name implies, it simulates the Scratch environment. The player has to select a character and add block instances to a stack of blocks that control the character's behavior on the stage. The figure shows the white stage and different sprites (buttons, arrows and a cartoon character), as well as the graphics output area. Blocks are code elements that control the behavior of the stage and sprites [13]. When a sprite is selected, its blocks element are shown in the Code panel. Figure 1 (center) shows the blocks that are connected to the cartoon cat, whose sprite is selected (e.g., blocks *Hide* and *Show*).

Inspired by the creativity test of Torrance [23], we measure creativity on three scales: fluency, flexibility, and originality. Generally speaking, fluency refers to the amount of ideas involved in the Scratch project, flexibility to the diversity of ideas, and originality to the distance between a project and typical projects [10]. In the following, we describe how we compute these scales for code and visual aspects, respectively. For both aspects, our strategy is to first define a distance between building components (e.g. code blocks and images) and then compute fluency, flexibility, and originality based on that distance.

Code Creativity. We represent the Scratch code as a collection of syntax trees, one representing the stage, and one for each sprite. Each syntax tree, in turn, consists of code blocks. Figure 2 shows a graph of the blocks in Scratch, where blocks are connected to the semantic sub-category they belong to (like *move* or *events*) and sub-categories are connected to categories, namely basic blocks, extension blocks, and custom blocks. Let now δ be the shortest-path distance between blocks in this graph. More specifically, we define δ as zero for equal blocks (e.g., two *Move* blocks), as 1 if the blocks are different but within the same sub-category of blocks (e.g., *Move*, *Turn*), and as 2 if the blocks are different and from different sub-categories (e.g., *Move*, *When Key Pressed*). To explain, we add one unit of distance for the transition between the sub-categories and another for the blocks being different.

For different categories, the distance between the pre-defined blocks and the extension blocks is defined as 3, and the distance between the pre-defined blocks to the custom blocks is defined 4. To explain, we add one one unit of distance for the block's difference, the second unit of distance due to the different sub-categories, and the third for the category change. Since the Scratch environment provides by default the pre-defined blocks, while custom blocks require the user to build something new, we add an additional unit of distance.

Based on the distance, we define code fluency of a Scratch project as the sum $\sum_x \delta(x, 0)$, where x are the code blocks in the project and 0 is the gray zero node in Figure 2. The distance $\delta(x, 0)$ to basic blocks (e.g. *Move*) is defined as 3, the distance $\delta(x, 0)$ to extension blocks (e.g., *Pen-Up*, *Language*) is 4, and the distance $\delta(x, 0)$ to custom blocks is 5. In other words, we assign higher fluency for the production of non-existent components or the use of custom blocks that require additional user effort. For example, in the 'Scratch in Scratch!' program the *Show* block presented in Figure 1 is a basic block from the sub-category 'Looks'. The program gets 3 points for this block, and an overall fluency score of 10523[2].

To compute code flexibility, we remove all duplicated blocks,

[1]`https://www.scratch-wiki.info/`

[2]For mathematical reasons we square the distances, thus yielding large numbers.

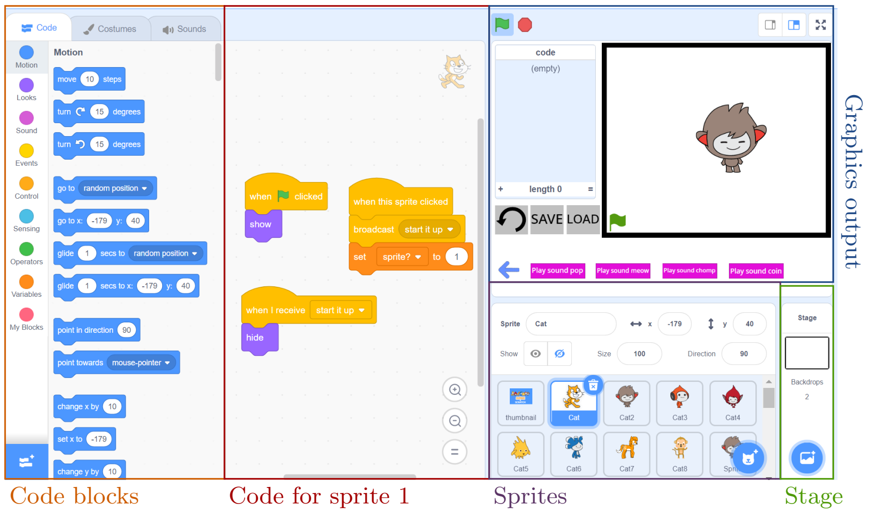


Figure 1: A screenshot of an example Scratch project. Left: A selection of possible code blocks. Center: The code blocks related to the currently selected sprite (the cat). Top right: The current graphical output. Bottom right: An overview of all sprites and the stage (i.e. the background).

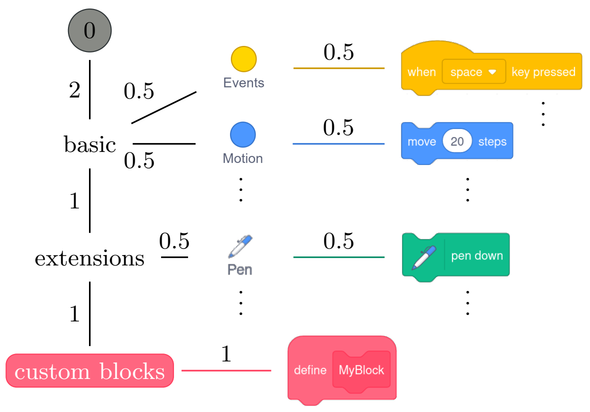


Figure 2: A graph of code blocks in scratch. We compute the distance between blocks by their shortest path distance in the graph, e.g. $\delta($move 20 steps, pen down$) = 0.5 + 0.5 + 1 + 0.5 + 0.5 = 3$.

then compute the sum of all pairwise distances $\sum_x \sum_y \delta(x, y)$ between code blocks in the project and divide by the number of unique code blocks. This measures how different the code blocks were, capturing the idea of flexibility as variability of concepts [10]. For example, the project 'Scratch in Scratch!' uses the blocks *Hide* and *Show* for each sprite. This increases fluency but not flexibility. Further, these two pre-defined blocks belong to the same sub-category 'Looks' and so $\delta(\mathit{Hide}, \mathit{Show}) = 1$. Overall, after normalizing by the number of the unique blocks in the program (58), it obtained a flexibility score of 395.37

We define code originality as the average distance of a Scratch project to a sample of typical projects in our data set, i.e. a project that is more distant from typical projects is considered more original. To compute the distance between projects, we follow the approach of Price et al. [22]. In particular, we use a three-step algorithm to construct an alignment between two Scratch projects. First, we compute the tree edit distance [26] between the stage syntax trees of both programs. Then, we compute all pairwise tree edit distances between sprites in both programs. Finally, we feed this result into the Hungarian algorithm [15] to obtain an alignment between the sprite trees. This is because sprites in a Scratch project do not have a clear ordering, making an unordered representation more natural.

While fluency and flexibility are based only on the program itself, the originality requires a reference sample of projects, i.e. we need a reference point with respect to which a project is original or not [23, 20]. To illustrate the effect of the reference set, we note that the originality of the 'Scratch in Scratch!' with respect to a reference set of 3 different project groups from the user study was 4488.84, 4168.89, and 2759, respectively.

Visual creativity. To represent visuals, we first collect all images (i.e. sprite and stage images) in our training data set and feed them into a ResNet50 neural network[3]. ResNet50 has been shown to generalize diverse image processing tasks and classifications [19, 16]. Accordingly, we hope that the representation of ResNet50 also helps to capture the semantic distance between images for our case. The output is a set of vectors, one for each image. To measure distance δ between images, we use the Cosine distance $\delta(x, y) = 1 - \frac{x^T \cdot y}{\|x\| \cdot \|y\|}$ because it is invariant against effects of scale/size, which would otherwise be a confounder in our data.

We compute visual fluency as the number of images in the Scratch projects, which is equivalent to the fluency definition of Torrance [23]. For example, in the project 'Scratch in

[3]`https://www.kaggle.com/keras/resnet50`

Scratch!' we have 47 images and that is its fluency score.

To measure visual flexibility, we use the same approach as for code flexibility, i.e. we compute the sum $\sum_x \sum_y \delta(x, y)$ of all pairwise distances over images in the project and then divide by the number of images. To illustrate, the program 'Scratch in Scratch' contains 2 similar button images 'Save' and 'Load' (see Figure 1). The Cosine distance between these images is relatively small $\delta('Save', 'Load') = 0.19$, based on the vectors created by the ResNet50 network. The flexibility score of the visual aspect of the project is 12.42.

We compute visual originality as the average distance of each image in a project to images in typical projects. To illustrate, the project 'Scratch in Scratch!' received an originality score of 0.57, 0.57 and 0.58 when using 3 different reference sets.

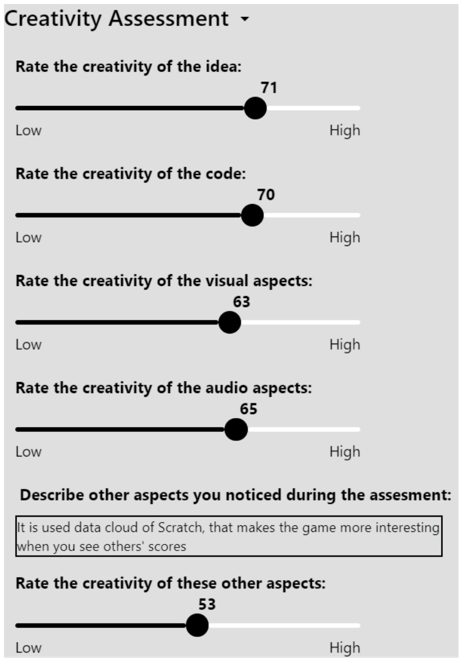
4. HUMAN CREATIVITY ASSESSMENT

In this section, we describe a user study to collect expert evaluations of creativity of Scratch projects. The experts were Scratch instructors without prior knowledge in creativity theory. Each expert was assigned a set of pre-selected Scratch projects and asked to separately evaluate the creativity of projects according to four different aspects: code, visual, audio and idea behind the project, which was identified in past work as an important factor in the creative process in Scratch [18].

We designed an online application to facilitate the rating process and to allow the experts to play and review each project as they see fit. The application was divided into three main screens. The Home Screen displays all of the projects that are assigned to an expert. When clicking on a project in the Home Screen, experts were able to see additional information about the project (e.g., the number of views and likes that the project received) and information about the user (e.g., country, date of registration in Scratch, and age if available) and also a link to the editor environment for the project's code and visuals and the embedded playable project.

For each project, experts were asked to answer questions that relate to the creativity of the four aspects of the Scratch project. Questions relating to visual aspects, such as whether the project contained images provided by scratch or originally created by the user. Experts were also asked to rate the novelty/quality and effort put into the visual aspects of the project. Questions relating to the project code, such as evaluating the code complexity, efficiency and novelty, and rating the effort put into the code. Questions about the project idea asked to include a short description of the idea and ratings for how much novelty and effort were required for developing the idea. If the project included sounds, experts were asked if these sounds were recorded by the user, imported or were provided by Scratch. Additionally, the experts rated the novelty of the sounds and the effort invested in the audio aspect.

Experts were also asked to provide a creativity score for each of the aspects (0-100), shown in Figure 3, as well as provide a weight (between 0 and 1, summing to 1) for each aspect according to its subjective importance in determining the creativity of a project. The creativity score of a given project for an expert is computed as the weighted summation of the creativity ratings for each aspect. The creativity score for each project is shown in the Home Screen, allowing experts to compare the scores and revise them at will.


Creativity Assessment
Rate the creativity of the idea:
71
Low
High
Rate the creativity of the code:
70
Low
High
Rate the creativity of the visual aspects:
63
Low
High
Rate the creativity of the audio aspects:
65
Low
High
Describe other aspects you noticed during the assesment:
Rate the creativity of these other aspects:
53
Low
High

It is used data cloud of Scratch, that makes the game more interesting when you see others' scores

Figure 3: Overall creativity assessment.

Table 1: Experts grading statistics of code, visual and final creativity scores

Expert	Statistic	Code	Visual	Final score
1	Mean	69.55	75.85	67.59
	SD	24.04	24.97	21.31
2	Mean	66.75	67.70	66.89
	SD	13.11	14.28	13.80
3	Mean	70.75	77.65	65.30
	SD	10.18	10.50	11.89
4	Mean	72.90	83.40	76.11
	SD	24.00	20.11	17.78
5	Mean	64.60	68.55	63.52
	SD	15.27	13.15	13.17

4.1 User Study

We recruited 5 experts from 4 countries: Cuba, Vietnam, India and Israel. All experts had at least two years of Scratch training experience to students of different ages in schools and after-school activities. We selected 45 unique projects of different types (games and stories), created by different users (age ranged between 9 to 18, from 25 different countries, and with different experience, from 4 to 258 projects). We uniformly sampled projects to each of the experts from this set, so that there is a sufficient spread of creativity assessments across project, while still having some projects being rated by several experts. Four of the experts evaluated 20 projects, while one evaluated 10 projects.

Table 1 presents the statistics of the scores for code, visuals, and the final creativity scores provided by the experts. We note that the highest scores were provided by expert 4 and that this expert as well as expert 1 had the highest standard deviation across all aspects. Experts 2 and 5, on the other hand, gave relatively lower scores with a lower standard deviation.

4.2 Agreement between experts

Experts differed widely in the creativity scores they assigned to projects. For example, experts 1, 2, and 3 all evaluated the project 'Scratch in Scratch!'. Expert 1 gave this project a creativity score of 91 for the code aspect, while expert 2 gave it a score of 67, and expert 3 gave a score of 82.

We note that the low agreement between raters should not signify a mistake or lack of expertise. It reflects the fact that creativity assessment is largely subjective, and that experts can differ about which aspects are more or less important when measuring creativity, as we show in this section. To compensate for this, we measure agreement using the Kendall Rank Correlation Coefficient [1]. This measure ranges from -1 (complete disagreement between rankings) to 1 (perfect match) and is determined based on overlapping projects for each pair of experts.

Figure 4 displays for each pair of experts the number of overlapping projects (in parentheses) and the Kendall-τ score. Note that experts 2 and 4 had only one overlapping project, therefore the Kendall-τ score cannot be calculated for them. As shown in the figure, the highest agreement (Kendall-$\tau = 0.67$) is between expert 5 and 2. The other positive agreement scores are much lower and vary between 0.2 and 0.33. Moreover, we see four pairs of experts with negative Kendall-τ scores, with 2 of them including expert 4.

The experts with the highest agreement score (experts 2 and 5) also exhibited similar scores for code and visual aspects (See Table 1), suggesting that they interpret creativity for these aspects in similar ways. However, the same expert 5 commonly disagreed with expert 1 (Kendall-$\tau = -0.6$). Their scores and rankings of overlapping projects differed substantially, suggesting that they differ in their interpretation of creativity. For example for the code aspect, the same project was ranked 1st by expert 5 and 16th by expert 1.

We observe that most experts found the visual aspect more significant than the code aspect when evaluating creativity. For the majority of experts, the project idea was the most important aspect. By contrast, experts assigned low weights to audio aspects. We note that the project idea is very difficult to model computationally. This is an interesting avenue to explore in future work.

5. PREDICTING CREATIVITY SCORES

In this section we report on the design and evaluation of a computational model to predict the creativity scores of Scratch projects. We build an automatic tool to support teachers (and students) in Scratch that can be trained on examples taken from individual or multiple experts.

We use an XGBoost Regressor [4] to predict the expert creativity scores for each project. As input features we used the

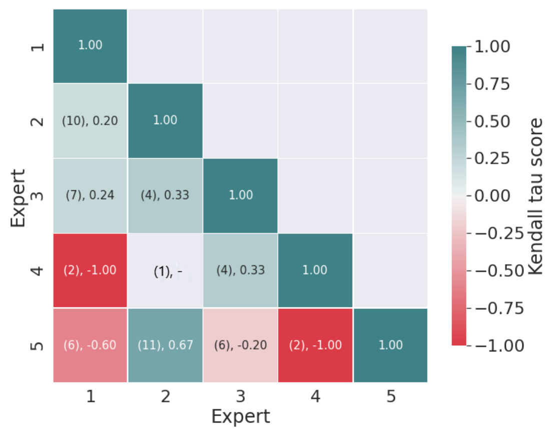


Figure 4: Kendall-τ Agreement between pairs of experts with overlapping projects on the final creativity score. (The number of overlapping projects is shown in parentheses.)

originality, flexibility, and fluency measures for both visual and code aspects, as described in Section 3. This provided us with 6 features for each instance (project). The reference sample for computing originality included all of the projects that the expert rated.

We created 2 types of XGBoost models: (1) a single rater model trained on projects for each expert separately and (2) a combined model trained on the projects from all experts together. Note that for the combined model, projects that were evaluated by more than one rater were treated as different instances. For each type of model, we created three different prediction models (a) predicting the code creativity score. (b) predicting the visual creativity score. (c) predicting the overall project score by the weighted combination score (visual and code). The features consisted of the originality, flexibility and fluency for the code aspect (model a), the visual aspect (model b), or both (model c).

The combined model and the single rater models were developed using the official implementation of XGBoost[4]. We selected the hyperparameters based on the structure of our data. We set the upper complexity limit of the model to six trees for the rater with 10 projects, 14 for the rest of the raters, and 29 trees for the combined raters and the maximum tree depth based on the number of features. The combined model was evaluated using 10-fold cross-validation. The single rater models were evaluated using 5 folds to ensure that the test set contained at least two projects.

Because of the high degree of variance between the raters, we do not seek to minimize error with respect to the predicted creativity score. Instead, we compare rankings. Ideally, we would compare the rankings of the projects in the test set with the true rankings for each expert. However, the size of the test set for some folds for some of the experts was small (4 projects for most experts). To increase the number of comparisons for Kendall-τ, we built a complete ranking

[4]https://github.com/dmlc/xgboost

Table 2: Kendall-τ score between XGBoost Regressor and experts scores - code creativity score, visual creativity score, and the weighed visual and code score

Experts	Kendall-τ		
Expert	Code	Visual	Weighed visual and code
1	0.52	0.52	0.42
2	0.51	0.42	0.42
3	0.53	0.58	0.36
4	0.46	0.52	0.57
5	0.52	0.42	0.50
Combined	0.43	0.44	0.42

over all projects for an expert, by combining the predicted scores of projects in the test-set with the scores of projects in the training set. However, we compute the Kendall-τ agreement only for project pairs with at least one project in the test set. We make a similar computation with respect to computing the Kendall-τ for the combined model.

Table 2 presents the Kendall-τ performance, computed as described above. As seen from the table, when predicting the creativity score for visual and code aspects, we achieve a Kendall-τ score of 0.51 and above for 3 out of 5 experts. When predicting the weighted creativity score, we achieve a Kendall-τ score of over 0.42 for 4 out of 5 experts. Overall, the agreement measure is higher than that of the inner-agreement between the experts themselves that is reported in Figure 4 (except for the pair 2 and 5).

The bottom row in Table 2 presents the results for the XGBoost model that is trained over the combined set for all experts. In all cases we achieve a Kendall-τ score above 0.42, which is higher than the inner-agreement scores for most pairs of experts. For visual aspects, the combined model is less successful than the individual models. In contrast, for the visual creativity, the combined model is better than the single rater model for two of the experts (experts 2 and 5); for weighted visual and code creativity score, the combined model is better or equal than the single rater model for 3 experts (experts 1, 2 and 3). This suggests that our models can define useful rules for aggregating creativity rankings by different experts despite the disagreement between them.

6. DISCUSSION AND CONCLUSION

In this paper we presented a formalization of creativity in terms of fluency, flexibility, and originality. We automatically computed creativity both for code and for visual aspects of Scratch projects and we intend to add the other possible modalities of that environment to our future work. Further, we set up a web application to rate the creativity in Scratch projects independently of our formalization. Finally, we recorded the ratings of five human experts on 45 Scratch projects via this application.

We observed that human raters tend to disagree on which projects are creative and which are not. Still, we were able to train regression forests, which achieved a higher ranking agreement with the human raters than they achieved with each other, and which only used the automatically generated ratings as input. We observed that the regression forest model could further improve its accuracy when being applied to individual experts instead of their shared data.

Our approach makes a step towards supporting teacher's abilities to detect and support creative outcomes in students' work. Ample future work is still to be done. Furthermore, we plan to analyze creativity ratings over time, thus tracking students' creative learning process. Future work will also need to address how an automatic assessment of creativity can support creativity at scale in technological environments, taking into account different subjective interpretations.

7. ACKNOWLEDGMENTS

This paper was made possible by funding from the German Research Foundation (DFG) under grant number PA 3460/2-1.

References

[1] H. Abdi. The Kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA*, 1:508–510, 2007.

[2] T. M. Amabile. *Creativity in context: Update to the social psychology of creativity.* Routledge, 2018.

[3] J. Bustillo and P. Garaizar. Using Scratch to foster creativity behind bars: Two positive experiences in jail. *Thinking Skills and Creativity*, 19:60 – 72, 2016.

[4] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proc. SIGKDD*, page 785–794, 2016.

[5] M. N. Giannakos, L. Jaccheri, and R. Proto. Teaching computer science to young children through creativity: Lessons learned from the case of Norway. In *Proc. CSERC*, page 103–111, 2013.

[6] J. P. Guilford. The structure of intellect. *Psychological bulletin*, 53(4):267–293, 1956.

[7] D. Henriksen, P. Mishra, and P. Fisser. Infusing creativity and technology in 21st century education: A systemic view for change. *Educational Technology & Society*, 19(3):27–37, 2016.

[8] A. Hershkovitz, R. Sitman, R. Israel-Fishelson, A. Eguíluz, P. Garaizar, and M. Guenaga. Creativity in the acquisition of computational thinking. *Interactive Learning Environments*, 27(5-6):628–644, 2019.

[9] R. Israel-Fishelson, A. Hershkovitz, A. Eguíluz, P. Garaizar, and M. Guenaga. The associations between computational thinking and creativity: The role of personal characteristics. *Journal of Educational Computing Research*, 58(8):1415–1447, 2021.

[10] K. H. Kim. Can we trust creativity tests? A review of the torrance tests of creative thinking (TTCT). *Creativity Research Journal*, 18(1):3–14, 2006.

[11] M. Knobelsdorf and R. Romeike. Creativity as a pathway to computer science. In *Proc. ITiCSE*, page 286–290, 2008.

[12] A. Kovalkov, A. Segal, and K. Gal. Inferring creativity in visual programming environments. In *Proc. L@S*, page 269–272, 2020.

[13] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education*, 10(4), Nov. 2010.

[14] D. R. Mullet, A. Willerson, K. N. Lamb, and T. Kettler. Examining teacher perceptions of creativity: A systematic review of the literature. *Thinking Skills and Creativity*, 21:9–30, 2016.

[15] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[16] A. S. B. Reddy and D. S. Juliet. Transfer learning with resnet-50 for malaria cell-image classification. In *Proceedings of the International Conference on Communication and Signal Processing (ICCSP 2019)*, pages 0945–0949, 2019.

[17] M. Resnick, K. Brennan, C. Cobo, and P. Schmidt. Creative learning @ scale. In *Proc. L@S*, page 99–100, 2017.

[18] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[19] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. de Geus. Malicious software classification using transfer learning of resnet-50 deep neural network. In *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA 2017)*, pages 1011–1014, 2017.

[20] M. A. Runco and G. J. Jaeger. The standard definition of creativity. *Creativity Research Journal*, 24(1):92–96, 2012.

[21] R. Shillo, N. Hoernle, and K. Gal. Detecting creativity in an open ended geometry environment. *International Educational Data Mining Society*, 2019.

[22] P. Thomas W., Z. Rui, and B. Tiffany. Evaluation of a data-driven feedback algorithm for open-ended programming. In *Proc. EDM*, pages 192–197, 2017.

[23] E. P. Torrance. Predictive validity of the torrance tests of creative thinking. *The Journal of creative behavior*, 6(4):236–252, 1972.

[24] S. Wheeler, S. Waite, and C. Bromfield. Promoting creative thinking through the use of ict. *Journal of Computer Assisted Learning*, 18(3):367–378, 2002.

[25] F. E. Williams. Assessing creativity across williams "cube" model. *Gifted Child Quarterly*, 23(4):748–756, 1979.

[26] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.