CODEMASTER: UM MODELO DE AVALIAÇÃO DO PENSAMENTO COMPUTACIONAL NA EDUCAÇÃO BÁSICA ATRAVÉS DA ANÁLISE DE CÓDIGO DE LINGUAGEM DE PROGRAMAÇÃO VISUAL

Nathalia da Cruz Alves – UFSC
Christiane Gresse von Wangenheim – UFSC
Jean Carlo Rossa Hauck – UFSC
Adriano Ferreti Borgatto – UFSC
Dalton Francisco de Andrade – UFSC

RESUMO

O ensino do pensamento computacional, já na Educação Básica, é de suma importância para preparar os alunos para os desafios do século XXI. Desta forma, surge a necessidade de avaliação das competências adquiridas em relação ao pensamento computacional. A avaliação pela análise do código criado pelo aluno como resultado de atividades abertas é uma abordagem que permite verificar quais conceitos foram efetivamente aplicados no processo de ensino-aprendizagem. O presente trabalho apresenta um modelo de avaliação de conceitos de algoritmos e programação como parte do pensamento computacional voltado a linguagens de programação baseado em blocos. O modelo foi sistematicamente desenvolvido com base em currículos de referência definindo uma rubrica e automatizado por um sistema web. Resultados de uma avaliação em larga escala com mais de 88 mil aplicativos criados com App Inventor, uma das linguagens baseada em blocos mais populares, indicam que o modelo é confiável (Alfa de Cronbach = 0,84). Os resultados também demonstram validade por meio de um grau satisfatório de correlação encontrado entre a maioria dos itens. A partir da disponibilidade do modelo, espera-se facilitar e reduzir o esforço necessário para avaliação de atividades de programação no ensino de computação na Educação Básica, suportando assim a sua ampla aplicação em escolas brasileiras.

INTRODUÇÃO

Atualmente, a computação encontra-se em evidência na maioria das atividades humanas. Independentemente da área de atuação de um profissional, é importante que este conheça os fundamentos e princípios básicos da computação para que possa exercer sua atividade de forma plena (CSTA, 2016). Assim, com objetivo de formar indivíduos preparados para os desafios do século XXI, propõe-se iniciar o ensino de computação já na Educação Básica. Em todos os lugares do mundo vários esforços estão sendo feitos neste sentido. Recentemente, no Brasil, em dezembro de 2018, o MEC homologou a BNCC - Base Nacional Comum Curricular – a qual apresenta o desenvolvimento do pensamento computacional dentro da área de Matemática (MEC, 2018).

> A área de Matemática, no Ensino Fundamental, centra-se na compreensão de conceitos e procedimentos em seus diferentes campos e no desenvolvimento do pensamento computacional, visando à resolução e formulação de problemas em contextos diversos. (MEC; BNCC, 2018, p. 471).

Sendo assim, professores do mundo todo criaram conjuntamente o *K-12 Computer Science Framework* (CSTA, 2016) que se trata de um guia de currículo de computação para a Educação Básica. Este guia define o pensamento computacional como um conjunto de práticas que podem ser abordadas via conceitos de algoritmos e programação. Na prática, o ensino de conceitos de computação é frequentemente feito por atividades de programação usando linguagens de programação visual baseada em blocos. Entre as linguagens existentes, App Inventor é uma das mais populares a qual permite qualquer indivíduo criar aplicativos para dispositivos Android.

Como parte do processo de ensino-aprendizagem, a avaliação é importante para fornecer feedback ao aluno e aos professores. Na prática, a avaliação da aprendizagem de pensamento computacional é tipicamente uma tarefa repetitiva e demorada requerendo esforço e tempo considerável, especialmente no contexto escolar com grandes turmas (INEP, 2015). Além disso, ela requer conhecimento substancial da computação, o que complica mais ainda a inserção do ensino de computação de forma interdisciplinar por professores de outras áreas. Uma forma de mitigar algumas destas dificuldades é usar uma ferramenta para auxiliar a avaliação de forma automatizada. Algumas vantagens, além da redução do esforço, são o aumento da consistência entre as avaliações e um retorno mais rápido (em tempo real). Isso libera também o professor para avaliar aspectos mais subjetivos, como por exemplo a criatividade.

Atualmente, já existem algumas abordagens automatizadas para avaliação do pensamento computacional voltadas à Educação Básica, como por exemplo, Dr. Scratch (MORENO-LEÓN; ROBLES, 2015), Ninja Code Village (OTA; MORIMOTO; KATO, 2016), entre outras. No entanto, a maioria destas abordagens não foi avaliada em relação à sua validade, confiabilidade e qualidade. Além disso, as abordagens não estão contextualizadas para a aplicação em escolas brasileiras. Assim, este trabalho apresenta um modelo, incluindo a definição sistemática de critérios de avaliação bem definidos a partir do *K-12 Computer Science Framework* (CSTA, 2016), bem como a Base Nacional Comum Curricular (MEC, 2018).

METODOLOGIA

O método de pesquisa utilizado neste trabalho é caracterizado como uma pesquisa aplicada de abordagem multimétodo (SAUNDERS et al., 2009). Possui natureza aplicada, haja vista

compreende o desenvolvimento de um modelo conceitual para análise de código. Tem abordagem multimétodo, pois utiliza técnicas qualitativas e quantitativas.

Com base no *K-12 Computer Science Framework* (CSTA, 2016), na Base Nacional Comum Curricular (MEC, 2018) e em outras referências relacionadas a linguagens baseada em blocos são definidos os objetivos de aprendizagem a serem avaliados. Com base nessa definição, critérios de avaliação são sistematicamente derivados para cada objetivo usando a abordagem GQM – *Goal Question Metric* (BASILI et al., 1994). A partir do modelo desenvolvido é instanciada uma rubrica para a linguagem de programação visual App Inventor. Visando a automatização da avaliação é evoluído um sistema web CodeMaster 1.0 (GRESSE VON WANGENHEIM et al., 2018) seguindo um processo iterativo incremental de desenvolvimento de software (LARMAN; BASILI, 2003), o qual define o nível de desempenho a partir da análise do código.

O modelo, quando desenvolvido, foi avaliado por um estudo de caso (YIN, 2017). Neste trabalho, para avaliar o modelo são coletados mais de 88 mil aplicativos reais da Galeria App Inventor, os quais são analisados com CodeMaster 2.0. A partir destes dados é realizada uma análise estatística sobre a confiabilidade do modelo por meio do coeficiente Alfa de Cronbach (CRONBACH, 1951) e validade por meio de uma análise de correlação (COHEN, 1998).

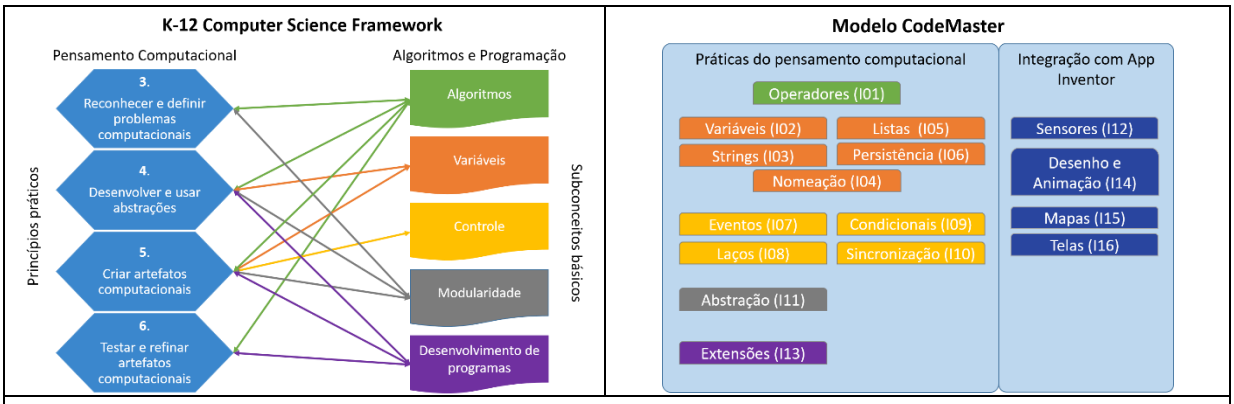
RESULTADOS E DISCUSSÃO

Modelo CodeMaster 2.0

O modelo CodeMaster 2.0 (ALVES, 2019) baseia-se na definição de que o pensamento computacional envolve a compreensão acerca da capacidade dos computadores, formulação de problemas e, principalmente, de algoritmos e programas que possam ser executados por um computador (CSTA, 2016). Dentro do contexto da Educação Básica, a BNCC também apresenta, em um nível abstrato, como parte da área de Matemática, algumas habilidades fortemente relacionadas às competências descritas pelo CSTA, as quais um aluno do Ensino Médio deve desenvolver (MEC, 2018). Estas competências do pensamento computacional, definidas pela BNCC e relacionadas ao *K-12 Computer Science Framework*, podem ser desenvolvidas por meio de unidades instrucionais que abordam conceitos de algoritmos e programação (Figura 1). Os conceitos de algoritmos e programação estão fortemente ligados aos princípios práticos 3-6 do *K-12 Computer Science Framework* (Figura 1).

O modelo CodeMaster 2.0 é dividido em duas subáreas do pensamento computacional (Figura 1). A subárea de princípios práticos, a qual contém os conceitos de algoritmos e programação, e a subárea de integração com a linguagem de programação visual baseada em blocos App Inventor que contém itens relacionados às características dessa linguagem.

Figura 1 – Detalhamento do modelo e da rubrica CodeMaster para App Inventor.



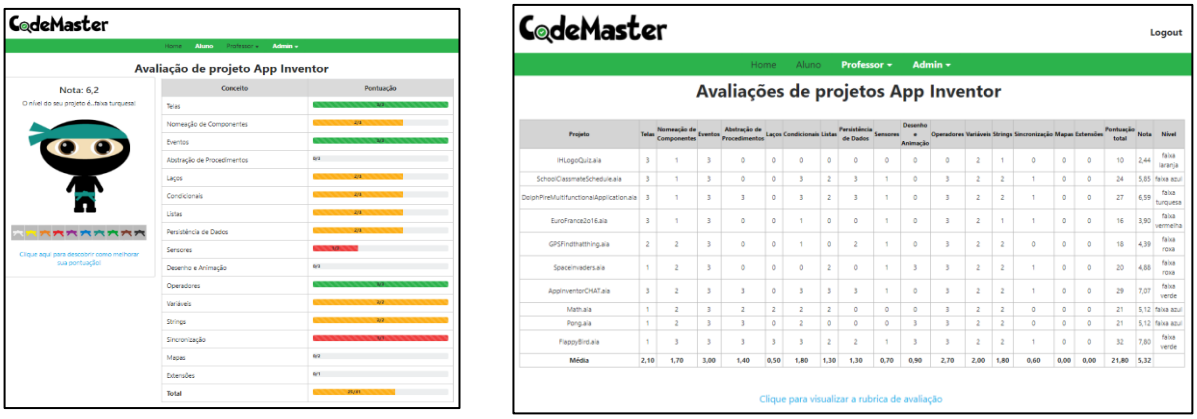
RUBRICA CODEMASTER 2.0 PARA APP INVENTOR				
PA1. Qual o nível de desempenho em algoritmos com relação às práticas do pensamento computacional?				
Operadores: Incorporar operadores (aritméticos, relacionais, booleanos) em programas.	**0 pontos**: Não usa operadores	**1 ponto**: Usa operadores aritméticos.	**2 pontos**: Usa operadores relacionais.	**3 pontos**: Usa operadores booleanos (lógicos).
PA2. Qual o nível de desempenho em representação de dados com relação às práticas do pensamento computacional?				
Variáveis: Criar programas com variáveis para armazenar, modificar dados e executar operações sobre seus valores.	**0 pontos**: Sem uso de variáveis.	**1 ponto**: Modificação ou uso de variáveis predefinidas.	**2 pontos**: Criação e operação com variáveis	
Strings: Criar programas com variáveis para armazenar, modificar dados e executar operações sobre seus valores.	**0 pontos**: Sem uso de *strings*.	**1 ponto**: Uso do comando de criação de *string* para alterar textos de elementos.	**2 pontos**: Criação e operação com *strings*.	
Nomeação: AP6: Criar variáveis com definição clara sem nomes genéricos ou default.	**0 pontos**: Nenhum ou poucos nomes são alterados do padrão. (menos do que 10%)	**1 ponto**: De 10 a 25% dos nomes são alterados do padrão.	**2 pontos**: De 26 a 75% dos nomes são alterados do padrão.	**3 pontos**: Mais de 76% dos nomes são alterados do padrão.
Listas: Usar listas para simplificar soluções, evitando o uso de variáveis simples repetidamente.	**0 pontos**: Não usa listas.	**1 ponto**: Usa uma lista unidimensional.	**2 pontos**: Usa mais de uma lista unidimensional.	**3 pontos**: Usa uma lista de tuplas (*map*).
Persistência de dados: Incorporar persistência de dados em programas.	**0 pontos**: Dados são armazenados em variáveis ou propriedades de componentes e não tem persistência quando app é fechado.	**1 ponto**: Usa persistência em arquivo (File ou Fusion Tables).	**2 pontos**: Usa algum dos bancos de dados locais do App Inventor (TinyDB).	**3 pontos**: Usa uma base de dados web TinyWebDB ou Firebase do App Inventor (Firebase ou TinyWebDB).
PA3. Qual o nível de desempenho em controle com relação às práticas do pensamento computacional?				
Eventos: Criar e desenvolver programas que incluam eventos	**0 pontos**: Nenhum manipulador de evento é usado (ex. *On click*).	**1 ponto**: 1 tipo de manipuladores de eventos é usado.	**2 pontos**: 2 tipos de manipuladores de eventos são usados.	**3 pontos**: Mais de 2 tipos de manipuladores de eventos são usados.
Laços: Criar e desenvolver programas que incluam laços	**0 pontos**: Não usa laços	**1 ponto**: Usa laço simples ("*While*")	**2 pontos**: Usa laço variável simples ("*For each*")	**3 pontos**: Usa laço item de lista ("*For each*")
Condicionais: Criar e desenvolver programas que incluam condicionais.	**0 pontos**: Não usa condicionais.	**1 ponto**: Usa apenas condicionais simples ("*if's*").	**2 pontos**: Usa condicionais completos ("*if then else*").	**3 pontos**: Usa um ou mais condicionais em sequência "*if - else if*".
Sincronização: Incorporar sincronização em programas.	**0 pontos**: Sem uso de temporizador para sincronização.	**1 ponto**: Uso de comando de temporizador para sincronização.		
PA4. Qual o nível de desempenho em modularidade com relação às práticas do pensamento computacional?				
Abstração: Decompor problemas em subproblemas para facilitar o desenvolvimento e revisão de	**0 pontos**: Não existem procedimentos.	**1 ponto**: Existe exatamente um	**2 pontos**: Existe mais de um procedimento.	**3 pontos**: Existem procedimentos tanto para organização quanto para

programas e criar procedimentos para organizar o código e torná-lo mais fácil de reutilizar		procedimento e sua chamada.		reuso. (Mais chamadas de procedimentos do que procedimentos).
PA5. Qual o nível de desempenho em desenvolvimento de programas com relação às práticas do pensamento computacional?				
Extensões: Incorporar código, mídia, biblioteca existente em programas originais e dar reconhecimento.	**0 pontos**: Sem uso de comandos de extensões.	**1 ponto**: Uso de comandos de extensões.		
PA6. Qual o nível de desempenho em integração entre elementos da linguagem com relação às práticas do pensamento computacional?				
Sensores: Incorporar elementos da VPL em programas	**0 pontos**: Sem uso de sensores.	**1 ponto**: Usa um tipo de sensor.	**2 pontos**: Usa 2 tipos de sensores.	**3 pontos**: Usa mais de 2 tipos de sensores.
Desenho e Animação: Incorporar elementos da VPL em programas	**0 pontos**: Sem uso de desenho e animação.	**1 ponto**: Uso de área sensível ao toque.	**2 pontos**: Uso de animação com bolinha predefinida.	**3 pontos**: Uso de animação com imagem.
Mapas: Incorporar elementos da VPL em programas	**0 pontos**: Sem uso de comandos de mapas.	**1 ponto**: Uso do comando de mapa.	**2 pontos**: Uso de comandos de marcadores de mapas.	
Telas: Incorporar elementos da VPL em programas	**0 pontos**: Apenas uma tela com componentes visuais e que seu estado não se altera com a execução do app (tela informativa).	**1 ponto**: Apenas uma tela com componentes visuais que se alteram com a execução do app.	**2 pontos**: Pelo menos duas telas e uma delas altera seu estado com a execução do app.	**3 pontos**: Duas ou mais telas e pelo menos 2 delas alteram seus estados com a execução do app.

Sistema Web CodeMaster 2.0

Com base na rubrica foi desenvolvido um suporte automatizando a avaliação via um sistema web. Nesse sistema, disponível *on-line* (http://apps.computacaonaescola.ufsc.br:8080/), o aluno pode individualmente avaliar o seu programa e/ou o professor todos os programas de uma turma inteira. A Figura 2 mostra como o CodeMaster 2.0 apresenta os resultados de uma avaliação individual ao aluno e de várias avaliações de uma turma ao professor.

Figura 2 – Interface apresentando a avaliação de um projeto ao aluno (à esquerda) e ao professor (à direita).



Avaliação do modelo CodeMaster 2.0

A confiabilidade do modelo para App Inventor foi analisada medindo a consistência interna por meio do coeficiente alfa de Cronbach (1951). Analisando os 15 itens do CodeMaster 2.0, obteve-se um alfa de Cronbach ($\alpha = 0{,}84$) considerado bom segundo DeVellis (2003). Complementando a análise do alfa de Cronbach, foi analisada a correlação item-total, bem como o valor do alfa de Cronbach (CRONBACH, 1951), excluindo-se o respectivo item (Tabela 1).

Tabela 1 – Resultados da análise da correlação item-total para a rubrica CodeMaster 2.0 para App Inventor.

Item	Correlação Item-total	Alfa de Cronbach se o item for removido
Operadores (I01)	0,694	0,82
Variáveis (I02)	0,686	0,82
Strings (I03)	0,583	0,83
Nomeação (I04)	0,585	0,82
Listas (I05)	0,364	0,84
Persistência (I06)	0,325	0,84
Eventos (I07)	0,596	0,82
Laços (I08)	0,286	0,84
Condicionais (I09)	0,618	0,82
Sincronização (I10)	0,562	0,83
Abstração (I11)	0,548	0,83
Sensores (I12)	0,448	0,84
Desenho e Animação (I14)	0,376	0,84
Mapas (I15)	0,015	0,85
Telas (I16)	0,324	0,84

Considera-se uma correlação satisfatória, se o coeficiente de correlação for maior que 0,29 (COHEN, 1998). Assim, nota-se que todos os itens, exceto mapas, apresentaram uma correlação muito próxima ou acima de 0,29, indicando que os itens têm uma correlação média ou alta com todos os outros itens. Além disso, nenhum item, quando excluído, causa um aumento no alfa de Cronbach, indicando que todos os itens contribuem para a validade da rubrica, com a exceção do item mapas, um recurso novo adicionado ao App Inventor e que, portanto, tem baixa representação no conjunto de dados que contém aplicativos mais antigos.

CONCLUSÕES

Neste artigo apresentamos um modelo para a avaliação do pensamento computacional via código-fonte. Os resultados obtidos com base numa avaliação de larga escala indicam boa confiabilidade e validade do modelo. Todos os itens, exceto um item relacionado a um recurso recentemente adicionado à linguagem de programação visual App Inventor (mapas), apresentaram bons indicadores, tanto de confiabilidade, como de validade. Com o resultado do presente trabalho disponibiliza-se um modelo de avaliação sistemática de programas criados com linguagens baseada em blocos contextualizado para a Educação Básica brasileira. Esse modelo pode suportar de forma eficiente e efetiva a avaliação no processo de aprendizagem tanto pelo aluno individualmente quanto pelo professor. E assim, contribuir positivamente para uma ampla aplicação do ensino de computação nas escolas Brasileiras.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e do Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil (CNPq).

REFERÊNCIAS

ALVES, N. da C. CodeMaster: Um Modelo de Avaliação do Pensamento Computacional na Educação Básica através da Análise de Código de Linguagem de Programação Visual. 2019. Dissertação – PPGCC/Universidade Federal de Santa Catarina.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. Goal Question Metric Paradigm. 2 ed. MARCINIAK, J. J.:Encyclopedia of Software Engineering, 1994, John Wiley & Sons.

COHEN, J. Statistical Power Analysis for the Behavioral Sciences. New York:Routledge Academic, 1998.

CRONBACH, L. J. Coefficient alpha and the internal structure of tests. Psychometrika, v. 16, n. 3, 1951.

CSTA. K-12 Computer Science Framework. Computer Science Teachers Association, 2016.

DEVELLIS, R. F. Scale development: theory and applications. Thousand Oaks: SAGE Publications, 2003.

GRESSE VON WANGENHEIM, C. et al. CodeMaster - Automatic Assessment and Grading of App Inventor and Snap! Programs. Informatics in Education, v. 17, n. 1, 2018. p. 117-150.

INEP. Indicadores Educacionais 2014. Brasília. 2015.

LARMAN, C.; BASILI, V. Iterative and incremental developments: a brief history. IEEE Computer, v. 36, 2003.

MEC, 2018. Base Nacional Comum Curricular, Brasil.

MORENO-LEÓN, J.; ROBLES, G. Dr. Scratch: a Web Tool to Automatically Evaluate Scratch Projects. Proceedings of the Workshop in Primary and Secondary Computing Education, Londres, Reino Unido, 2015.

OTA, G. et al. Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. IEEE Symposium on Visual Languages and Human-Centric Computing, Cambridge, UK, 2016. p. 238-239.

SAUNDERS, M. N. K., LEWIS, P.; THORNHILL, A. Research Methods for Business Students. 5 ed. Harlow:Prentice Hall, 2009.

YIN, R. K. Case study research: design and methods. (6th ed.) Thousand Oaks: SAGE Publications, 2017.