Um Modelo de Avaliação do Pensamento Computacional na Educação Básica através da Análise de Código de Linguagem de Programação Visual

Nathalia da C. Alves[1], Christiane G. von Wangenheim[1], Jean C. R. Hauck[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
88.049-200 – Florianópolis – SC – Brazil

nathalia.alves@posgrad.ufsc.br, {c.wangenheim,jean.hauck}@ufsc.br

Abstract. *Teaching computational thinking in K-12 is important to prepare students for the challenges of the 21st century. Consequently, there is a need to assess the acquired competences. In this context we present a model for the assessment of computational thinking competences related to algorithm and programming with visual programming languages. The model was systematically developed based on reference curricula and instantiated and automated for projects created with App Inventor. Results from a large-scale evaluation, with more than 88,000 mobile applications created with App Inventor, indicate good reliability and demonstrate the model's validity.*

Resumo. *O ensino do pensamento computacional na Educação Básica, é importante para preparar os alunos para os desafios do século XXI. Desta forma, surge a necessidade de avaliação das competências adquiridas. O presente trabalho apresenta um modelo de avaliação de conceitos de algoritmos e programação, como parte do pensamento computacional, com linguagens de programação visual. O modelo foi sistematicamente desenvolvido com base em currículos de referência e foi instanciado e automatizado para projetos criados com App Inventor. Resultados de uma avaliação em larga escala, com mais de 88 mil aplicativos criados com App Inventor, indicam boa confiabilidade e demonstram a validade do modelo.*

1. Introdução

Atualmente a computação encontra-se em evidência na maioria das atividades. Independentemente da área de atuação de um profissional, é importante que se conheça os fundamentos e princípios básicos da computação para que se possa exercer sua atividade de forma plena. A inserção da computação já na Educação Básica é vista como uma das abordagens mais efetivas neste contexto. Vários esforços estão sendo feitos nesse sentido mundialmente. Recentemente, no Brasil, o MEC homologou a Base Nacional Comum Curricular (BNCC) que apresenta o desenvolvimento do **pensamento computacional** dentro da área de Matemática [MEC 2018]. O pensamento computacional refere-se aos processos de pensamento envolvidos na criação de soluções algorítmicas, ou passo-a-passo, que podem ser executadas por um computador [Wing 2006]. Atualmente existem diversas formas de desenvolver o pensamento computacional, inclusive no Brasil [Santos et al. 2018].

A maioria dos estudos aborda o pensamento computacional por meio da aprendizagem baseada em problemas usando uma linguagem de programação visual (*Visual Programming Language*), como Scratch. Uma VPL é qualquer linguagem de

DOI: 10.5753/cbie.wcbie.2019.1021

programação que permite ao usuário criar programas por meio da manipulação de elementos gráficos em vez de especificá-los textualmente [Golin e Reiss 1990]. As VPLs são indicadas como sendo mais adequadas para o ensino de computação de novatos, pois permite aos alunos concentrarem-se apenas na programação lógica, sem o obstáculo de erros de sintaxe que são encontrados em linguagens tradicionais baseadas em texto [CSTA 2016]. Assim, várias iniciativas focam também no uso do App Inventor desenvolvendo apps, dada a proximidade dos alunos com *smartphones* (IBGE, 2015). App Inventor é um projeto de código aberto criado originalmente pela Google, atualmente é mantido pelo *Massachusetts Institute of Technology* (MIT) sendo usado por mais de 8 milhões de usuários no mundo inteiro.

Os programas resultantes das atividades práticas de uma unidade instrucional devem ser avaliados observando a aplicação de conceitos do pensamento computacional. [CSTA 2016]. No entanto, apesar dos esforços para abordar a avaliação do pensamento computacional [Grover e Pea 2013], atualmente não há consenso sobre quais estratégias devem ser utilizadas na avaliação [Grover et al. 2014] [Brennan e Resnick 2012]. Devido à natureza abstrata do objeto a ser medido, a avaliação do pensamento computacional é particularmente complexa [Yadav et al. 2015]. Diferentes abordagens e métodos foram propostos, no entanto, para avaliar atividades abertas e complexas, como parte da aprendizagem baseada em problemas, avaliações autênticas parecem um meio mais apropriado do que as avaliações tradicionais [Torrance 1995].

A avaliação autêntica, especificamente a avaliação baseada no desempenho, mede as competências exigindo tipicamente que os alunos trabalhem em uma tarefa aberta, como a programação de um artefato de software. Assim, indicadores relacionados aos objetivos de aprendizagem são definidos para avaliar o desempenho dos alunos por meio do produto. No contexto da avaliação do pensamento computacional, tais indicadores orientados para o resultado assumem que atributos mensuráveis podem ser extraídos do artefato de software criado pelo aluno, tais como: corretude, complexidade, conformidade com padrões de codificação, entre outros.

As definições de níveis de desempenho, baseados em critérios e indicadores sobre os resultados de aprendizagem, são tipicamente feitas por meio de rubricas [McCauley 2003]. Quando usadas para avaliar as atividades de programação, as rubricas mapeiam a capacidade de desenvolver um artefato de software para uma pontuação, indiretamente inferindo competências do pensamento computacional [Brennan e Resnick 2012] [Sherman e Martin 2015]. Exemplos incluem o Dr. Scratch [Moreno-León e Robles 2015], NinjaCodeVillage [Ota et al. 2016] ou a rubrica de pensamento computacional móvel [Sherman e Martin 2015]. Para apoiar a aplicação na prática, algumas dessas abordagens foram automatizadas, como o Dr. Scratch [Moreno-León e Robles 2015] ou o Ninja Code Village [Ota et al. 2016]. Essas ferramentas medem a complexidade do software realizando uma análise estática de código, contando o tipo e o número de blocos de comando usados, quantificando conceitos e práticas do pensamento computacional, como lógica, abstração, fluxo de controle etc. No entanto, a maioria dessas rubricas e ferramentas avaliam o pensamento computacional somente em Scratch [Alves et al. 2019].

A pesquisa sobre a avaliação de código criado com outras VPLs baseadas em blocos, incluindo o App Inventor, ainda é escassa. Sherman e Martin (2015) propõem uma rubrica de avaliação com o objetivo de avaliar o pensamento computacional móvel

por meio da análise de programas criados com App Inventor. Essa rubrica foi projetada para avaliar padrões de pensamento computacional móvel, incluindo seis critérios sobre pensamento computacional em geral e oito critérios específicos sobre pensamento computacional móvel, considerando que os aplicativos para *smartphones* abordam novos conceitos de computação relacionados à computação móvel. Uma avaliação da rubrica com base na análise de 45 aplicativos de 18 alunos do ensino superior demonstra a eficácia da rubrica para avaliar diferentes graus de competências do pensamento computacional. No entanto, a rubrica foi desenvolvida para o App Inventor Classic, o qual foi descontinuado em 2015 e substituído pelo App Inventor 2.0, incluindo novos recursos não cobertos por essa rubrica.

Assim, com base na literatura, o objetivo deste trabalho é propor um modelo de avaliação baseado no desempenho. Este modelo é instanciado por meio de uma rubrica de avaliação. A avaliação com base nessa rubrica é automatizada por meio da ferramenta web gratuita CodeMaster, evoluindo o modelo do estado da arte. A ferramenta faz a análise do código de programas criados com VPL baseada em blocos e fornece *feedback* aos alunos e professores na forma de uma pontuação do pensamento computacional. Uma avaliação preliminar conduzindo um teste de usabilidade com professores e alunos da Educação Básica indicou que a ferramenta é útil, utilizável e eficiente para apoiar a avaliação de projetos do App Inventor [Gresse von Wangenheim et al. 2018].

2. Método de pesquisa

O presente trabalho é caracterizado como uma pesquisa exploratória, pois tem como objetivo o aprimoramento de ideias ou a descoberta de novas hipóteses a partir da análise do estado da arte e de estudos de caso. Possui natureza aplicada, haja vista que compreende o desenvolvimento de um modelo conceitual para análise estática de código. Tem abordagem multimétodo pois utiliza técnicas qualitativas e quantitativas. As etapas dessa pesquisa incluem:

Levantamento do estado da arte. Nesta etapa é realizado um mapeamento sistemático de literatura [Petersen et al. 2008] com o objetivo de levantar o estado da arte sobre modelos existentes de análise automatizada de atividades de programação no contexto da Educação Básica. Inicialmente é definido o protocolo de busca. Usando os critérios de inclusão/exclusão, trabalhos relevantes são selecionados durante a execução da busca. Na etapa de análise dos resultados, são extraídas informações dos trabalhos selecionados e é realizada uma análise sobre pontos fortes e fracos dos trabalhos. O resultado do levantamento é publicado em [Alves et al. 2019].

Avaliação em larga escala de um modelo do estado da arte: Identificado na revisão da literatura como um dos principais modelos existentes relacionados ao escopo deste trabalho, é feita uma avaliação em larga escala do modelo CodeMaster 1.0 [Gresse von Wangenheim et al. 2018]. Essa avaliação é feita por meio de um estudo de caso [Yin 2001] com aplicativos coletados da Galeria App Inventor, os quais são analisados com a ferramenta CodeMaster 1.0. A partir dos dados de avaliação, é realizada uma análise sobre a confiabilidade [Cronbach 1951] e a validade, por meio de uma análise fatorial [Glorfeld 1995].

Desenvolvimento do modelo. O modelo é desenvolvido seguindo a abordagem de design instrucional ADDIE [Branch 2009]. Com base no *framework* CSTA (2016) são definidos os objetivos de aprendizagem e desenvolvidos critérios de avaliação para cada

objetivo. A partir do modelo desenvolvido é instanciada uma rubrica para App Inventor levando-se em consideração os resultados do mapeamento sistemático e da avaliação do modelo CodeMaster 1.0. A partir da rubrica instanciada, é completado o suporte automatizado no CodeMaster para suportar a nova rubrica definida. A implementação da rubrica segue um processo iterativo e incremental [Larman e Basili 2003] incluindo análise de requisitos, modelagem, construção de software e testes.

Avaliação em larga escala do modelo CodeMaster 2.0. O modelo desenvolvido é avaliado por um estudo de caso [Yin, 2001]. A partir do método GQM [Basili et al. 1994] é definido o que deve ser avaliado e quais instrumentos de coleta de dados são utilizados. Para avaliar o modelo são coletados aplicativos da Galeria AppInventor, os quais são analisados com o avaliador de código CodeMaster 2.0. A partir desses dados é realizada uma análise estatística sobre a confiabilidade do modelo por meio do coeficiente Alfa de Cronbach [Cronbach 1951]. Além disso, a validade é analisada por meio de uma análise fatorial exploratória [Glorfeld 1995]. Os dados são discutidos e interpretados de forma a identificar a confiabilidade e validade do modelo.

3. O modelo de avaliação CodeMaster 2.0

O modelo genérico CodeMaster 2.0 é um modelo para avaliar competências do pensamento computacional relacionadas a algoritmos e programação [CSTA 2017]. Concentra-se na avaliação de tarefas de programação abertas e complexas (*ill-structured*) as quais podem ter mais de uma solução correta, por exemplo, estudantes desenvolvendo seus próprios aplicativos para resolver um problema da comunidade. Seguindo a definição do CSTA (2016), o pensamento computacional é avaliado pelo modelo com base nas práticas 3–6 do K-12 *Computer Science Framework* (Figura 1). Cada prática do pensamento computacional é fortemente relacionada com os conceitos de algoritmos e programação [CSTA 2017] presentes também nas VPLs.

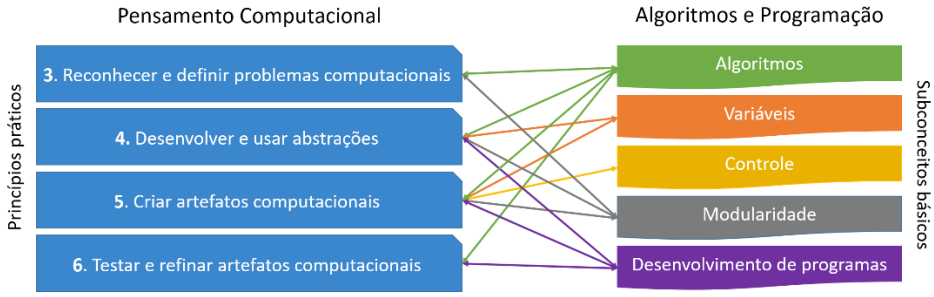

Figura 1. Relação entre as práticas do pensamento computacional e o conceito (e subconceitos) de algoritmos e programação segundo o CSTA (2016; 2017).

Levando em consideração a importância de se inserir o ensino de computação desde a Educação Básica e os avanços já realizados nesse sentido no Brasil a partir da BNCC [MEC, 2018] e no mundo [CSTA 2016], o modelo desenvolvido está inserido no contexto da Educação Básica, especificamente, a partir do 3º ano do Ensino Fundamental até a 2ª série do Ensino Médio, para alunos entre 8 e 16 anos.

Para a instanciação de uma rubrica do modelo é escolhido o ambiente App Inventor. App Inventor é um ambiente de programação com uma VPL baseada em blocos que permite a criação de aplicativos móveis para dispositivos Android. O modelo CodeMaster 2.0 para App Inventor adota uma estratégia de avaliação autêntica, medindo o desempenho dos alunos com base nos artefatos de software criados como resultados de

tarefas de programação. É baseado em uma rubrica que mede se indicadores de aprendizagem foram atingidos a fim de avaliar o grau de competência do pensamento computacional demonstrado pelo programa criado pelos alunos (Tabela 1).

Tabela 1. Detalhamento da rubrica CodeMaster 2.0 para App Inventor.

Item	0 pontos	1 ponto	2 pontos	3 pontos
Operadores	Não usa operadores	Usa operadores aritméticos.	Usa operadores relacionais.	Usa operadores booleanos (lógicos).
Variáveis	Sem uso de variáveis.	Modificação ou uso de variáveis predefinidas.	Criação e operação com variáveis	-
Strings	Sem uso de strings.	Uso de string para alterar textos de componentes.	Criação e operação com strings.	-
Nomeação	Menos do que 10% dos nomes são alterados do padrão.	10 a 25% dos nomes são alterados do padrão.	De 26 a 75% dos nomes são alterados do padrão.	Mais de 76% dos nomes são alterados do padrão.
Listas	Não usa listas.	Usa uma lista unidimensional.	Usa mais de uma lista unidimensional.	Usa uma lista de tuplas (map).
Persistência de dados	Não há persistência quando o app é fechado.	Usa persistência em arquivo.	Usa um banco de dados local do App Inventor.	Usa uma base de dados web.
Eventos	Nenhum manipulador de evento é usado (Ex.: On click).	1 tipo de manipulador de eventos é usado	2 tipos de manipuladores de eventos são usados	Mais de 2 tipos de manipuladores de eventos são usados
Laços	Não usa laços	Usa "While" (laço simples)	Usa "For each" (variável simples)	Usa "For each" (item de lista)
Condicionais	Não usa condicionais.	Usa apenas "if's" simples.	Usa apenas "if then else".	Usa um ou mais "if - else if".
Sincronização	Sem uso de temporizador para sincronização.	Uso de temporizador para sincronização.	-	-
Abstração	Não existem procedimentos.	Existe exatamente um procedimento e sua chamada.	Existe mais de um procedimento.	Existem procedimentos tanto para organização quanto para reuso.
Extensões	Sem uso de comandos de extensões.	Uso de comandos de extensões	-	-
Sensores	Sem uso de sensores.	Usa um tipo de sensor.	Usa 2 tipos de sensores.	Usa mais de 2 tipos de sensores
Desenho e Animação	Sem uso de desenho e animação	Uso de área sensível ao toque.	Uso de animação com bolinha predefinida.	Uso de animação com imagem.
Mapas	Sem uso de comandos de mapas.	Uso do comando de mapa.	Uso de comandos de marcadores de mapas.	-
Telas	Apenas uma tela com componentes visuais e que seu estado não se altera com a execução do app.	Apenas uma tela com componentes visuais que se alteram com a execução do app.	Pelo menos duas telas e uma delas altera seu estado com a execução do app.	Duas ou mais telas e pelo menos 2 delas alteram seus estados com a execução do app.

Os itens da rubrica CodeMaster 2.0 são definidos em uma escala ordinal, com níveis crescentes representando maior complexidade em relação ao conceito que está sendo medido (Tabela 1). Para cada nível de desempenho, são descritos comportamentos observáveis, variando de "item não está (ou minimamente) presente" a uma especificação de uso avançado do item. Como resultado, uma pontuação é atribuída para cada item. Uma nota final para um projeto, em relação ao pensamento computacional, é calculada por meio da soma das pontuações parciais, a qual varia de 0 a 41 pontos.

A rubrica foi desenvolvida com base na rubrica do Dr. Scratch, medindo a complexidade dos programas dos alunos em relação ao pensamento computacional [CSTA, 2016] por meio da quantificação dos comandos utilizados [Moreno-León e Robles 2015]. No entanto, nenhum critério de paralelismo é incluído, pois não é aplicável a programas do App Inventor [Turbak et al. 2014]. As diferenças relacionadas à rubrica de avaliação do pensamento computacional móvel [Sherman e Martin 2015] devem-se aos novos recursos disponíveis no App Inventor 2.0 que não estavam disponíveis quando aquela foi criada. Concentrando-se exclusivamente na avaliação baseada nos artefatos de

software criados, critérios relacionados ao processo de desenvolvimento e perspectivas do pensamento computacional, como proposto por Resnick (2012), também não são considerados.

A avaliação e a classificação dos projetos do App Inventor são automatizadas por meio de um sistema web gratuito com base na análise estática do código criado pelo aluno. Os alunos podem usar a ferramenta durante todo o processo de aprendizado para obter *feedback* imediato sobre o projeto. Os professores podem usar a ferramenta para avaliar e classificar os projetos de uma turma inteira como parte de uma avaliação abrangente, conforme sugerido por Brennan e Resnick (2012).

4. Avaliação do CodeMaster 2.0

O objetivo da avalição é analisar o modelo em termos de confiabilidade e validade por meio da instanciação da rubrica CodeMaster 2.0 para App Inventor. Para maximizar o tamanho da amostra, foi realizado o *download* de todos os aplicativos públicos disponíveis e acessíveis da Galeria do App Inventor até maio de 2018. Como resultado, foi analisado o código-fonte de 88.606 aplicativos App Inventor.

4.1. Existe evidência de consistência interna da rubrica CodeMaster 2.0?

Foi analisada a confiabilidade medindo a consistência interna da rubrica CodeMaster 2.0 para App Inventor por meio do coeficiente alfa de Cronbach (1951). Valores de alfa de Cronbach entre $0,7 < \alpha \leq 0,8$ são aceitáveis, entre $0,8 < \alpha \leq 0,9$ são bons e $\alpha \geq 0,9$ são excelentes [DeVellis 2003], indicando assim uma consistência interna do instrumento. Analisando os 15 itens da rubrica CodeMaster 2.0 (com o item Extensões excluído desta análise pois a Galeria do App Inventor não permite aplicativos com extensões), obteve-se um valor bom do alfa de Cronbach ($\alpha = 0,84$) e acima do valor alfa de Cronbach da rubrica CodeMaster 1.0 ($\alpha = 0,79$).

4.2. Existe evidência de validade convergente da rubrica CodeMaster 2.0?

Para obter evidências da validade convergente dos itens da rubrica, são calculadas as correlações com os demais itens da rubrica. Cada item deve ter uma correlação média ou alta com todos os outros itens [DeVellis 2003]. Além disso, é analisado como fica o alfa de Cronbach se um item for excluído. Os resultados desta análise são apresentados na Tabela 2 com o item 13 (Extensões) excluído desta análise pois a Galeria do App Inventor não permite aplicativos com extensões.

Tabela 2. Resultados da análise de validade para a rubrica CodeMaster 2.0.

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

Considera-se uma correlação satisfatória, se o coeficiente de correlação for maior que 0,29 [Cohen 1998]. Espera-se que nenhum item cause um aumento substancial no alfa de Cronbach se excluído [DeVellis 2003], assim, indicando que todos os itens contribuem para a validade da rubrica. A maioria dos valores das correlações item-total está acima de 0,29, indicando que existe uma boa consistência interna (Tabela 2). O grau de correlação entre os itens mostra que os itens medem o mesmo conceito, indicando evidência de validade convergente. A baixa correlação do item Mapas (item 15) pode ser explicada pelo fato de Mapas ser um recurso novo e ainda pouco usado e, portanto, sub-representado no conjunto de dados que possui aplicativos mais antigos. O I08 Laços apresenta uma correlação de 0,28, no entanto, a correlação ainda é muito próxima do valor mínimo esperado (0,29). Isso pode ser devido ao fato de que o uso de comandos de laços nos programas App Inventor é pequeno, pois muitos processos iterativos que seriam expressos com laços em outras linguagens de programação, são expressos como um evento que executa uma única etapa da iteração toda vez que é acionado [Turbak et al. 2014]. Todos os outros itens mostram uma diminuição no valor do alfa de Cronbach se removidos e demonstram correlação item-total suficiente (Tabela 2), indicando assim a validade das competências mensuradas relacionadas ao pensamento computacional.

4.3. Como os fatores subjacentes influenciam as respostas nos itens da rubrica?

Para analisar se os critérios da rubrica CodeMaster 2.0 para App Inventor podem ser submetidos a um processo de análise fatorial, foi utilizado o índice KMO [Brown, 2006] que mede a adequação da amostragem com valores entre 0 e 1. Um valor próximo a 1,0 suporta uma análise fatorial e qualquer valor menor que 0,5 indica que provavelmente não é passível de análise fatorial [Brown, 2006]. Analisando o conjunto de critérios da rubrica CodeMaster 2.0 para App Inventor, foi obtido um índice KMO de 0,83 indicando que a análise fatorial é apropriada neste caso. Assim, foi realizada uma análise fatorial para identificar o número de fatores que representa os itens da rubrica CodeMaster 2.0 (excluindo-se o item 13 pois a Galeria do App Inventor não permite aplicativos com extensões). Foi utilizada a análise paralela como método para determinar o número de fatores a serem retidos na análise fatorial [Glorfeld 1995]. Os resultados mostram que há um fator preponderante, mas ainda indica três pontos acima da linha vermelha (Figura 2).

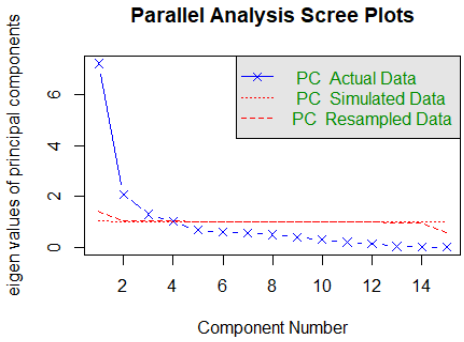

Figura 2. Scree Plot referente à rubrica CodeMaster 2.0 para App Inventor.

O scree plot sugere que pode haver 3 fatores subjacentes. Para determinar quais itens são carregados em qual fator, utilizou-se o método de rotação Oblimin, no qual os fatores podem ser correlacionados [Jackson 2005]. O limiar para as cargas fatoriais é baseado no de Comrey e Lee (1992) que sugerem os limites 0,32 (ruim), 0,45 (aceitável), 0,55 (bom), 0,63 (muito bom) ou 0,71 (excelente). Cargas fatoriais boas, acima de 0,55, estão em negrito na Tabela 3.

Tabela 3. Cargas para 3 fatores na rubrica CodeMaster 2.0 para App Inventor.

Item	Fator 1	Fator 2	Fator 3
I01. Operadores	0,325	0,074	**0,795**
I02. Variáveis	0,453	0,337	**0,763**
I03. Strings	-0,028	-0,100	**0,801**
I04. Nomeação	0,198	0,174	**0,659**
I05. Listas	-0,070	0,123	**0,690**
I06. Persistência	-0,093	-0,156	**0,786**
I07. Eventos	0,178	-0,157	**0,868**
I08. Laços	-0,004	0,209	**0,768**
I09. Condicionais	0,150	0,048	**0,807**
I10. Sincronização	**0,710**	-0,336	**0,616**
I11. Abstração	0,406	0,241	**0,779**
I12. Sensores	**0,713**	-0,432	0,453
I14. Desenho e Animação	**0,752**	0,298	0,351
I15. Mapas	-0,123	-0,389	0,403
I16. Telas	-0,158	-0,339	**0,702**

Analisando as cargas fatoriais dos itens (Tabela 3), pode-se observar que o fator 1 está mais relacionado aos conceitos de computação móvel abordados pelo App Inventor, com o I12 (Sensores) e o I14 (Desenho e Animação), ambos relacionados a este conceito sendo agrupados nesse fator. Apesar de o I10 (Sincronização) apresentar uma alta carga fatorial nesse primeiro fator, também apresenta uma alta carga fatorial no terceiro fator.

O segundo fator apresenta baixas cargas fatoriais para todos os itens com uma carga fatorial de todos os itens abaixo de 0,5, indicando que não existe este segundo fator.

O terceiro fator está mais relacionado ao pensamento computacional conforme definido pelo CSTA (2016). A maioria dos itens da rubrica apresenta uma alta carga fatorial nesse fator. Apesar de o I15 (Mapas), semanticamente relacionado ao primeiro fator, apresentar maior fator de carregamento no terceiro fator, esse item pode estar sub-representado no conjunto de dados, causando ruído no cálculo de seus indicadores.

4.4. Discussão

Os resultados obtidos indicam boa confiabilidade e validade da rubrica CodeMaster 2.0 para a avaliação das competências de pensamento computacional. Além disso, os itens também apresentaram resultados consistentes em relação às subdimensões da rubrica, e a análise fatorial mostrou que os itens possuem altas cargas fatoriais em duas subdimensões (Figura 3) conforme originalmente proposto. Apesar de o item 15 (Mapas) ter apresentado indicadores inadequados, cabe ressaltar que é um recurso adicionado recentemente ao App Inventor. Assim, muitos dos aplicativos utilizados na avaliação da rubrica, por serem mais antigos, não contêm esse recurso. Em relação aos demais itens, todos apresentaram indicadores melhores dos que foram obtidos na avaliação da rubrica CodeMaster 1.0.

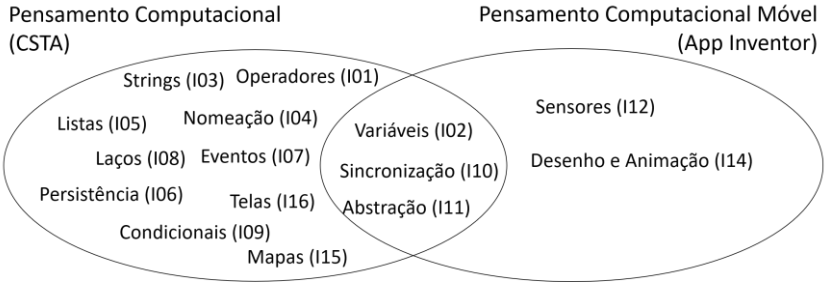

Figura 3. Subdimensões dos itens rubrica CodeMaster 2.0 para App Inventor.

Ameaças à validade. Ameaças relacionadas ao projeto do estudo de caso foram mitigadas por meio da definição e documentação de uma metodologia sistemática para o

estudo usando a abordagem GQM [Basili et al. 1994]. O risco referente à qualidade dos dados em termos de padronização é minimizado com todas as análises sendo realizadas de forma automatizada usando a mesma rubrica. O conjunto de dados vem de diversos contextos da comunidade App Inventor, de todos os lugares no mundo, e nenhuma informação adicional sobre o histórico dos criadores dos programas está disponível. No entanto, como o objetivo é a análise da validade da rubrica de forma independente do contexto, isso não apresenta um problema. Em termos de validade externa a análise é baseada em 88.606 aplicativos coletados da Galeria App Inventor, um tamanho de amostra satisfatório que permite a geração de resultados significativos.

5. Conclusão

Como parte do presente trabalho, foi analisado o estado da arte por meio de um mapeamento sistemático [Alves et al. 2019] e analisada a confiabilidade e validade de um modelo no estado da arte [Gresse von Wangenheim et al. 2018]. Como resultado foi criado um modelo confiável e válido para a avaliação de programas de VPLs criados como resultado de atividades abertas no contexto da Educação Básica, instanciado e automatizado para projetos de App Inventor. Trabalhos futuros incluem a avaliação de outros conceitos e a realização de estudos empíricos aplicando o modelo em escolas Brasileiras.

Referências

Alves, N. da C., von Wangenheim, C. G., Hauck, J. (2019) "Approaches to Assess Computational Thinking Competences Based on Code Analysis in K-12 Education: A Systematic Mapping Study", Informatics in Education, v. 18, p. 17-39.

Basili, V. R., Caldiera, G., Rombach, H. D. (1994) "Goal Question Metric Paradigm". 2 ed. Marciniak, J. J.:Encyclopedia of Software Engineering, John Wiley & Sons.

Branch, R. M. (2009) "Instructional Design: The ADDIE Approach", NY: Springer.

Brennan, K.; Resnick, M. (2012) "New frameworks for studying and assessing the development of computational thinking", Proc. of the Annual Meeting of the American Educational Research Association, Vancouver, Canada.

Brown, T. A. (2006) "Confirmatory factor analysis for applied research", NY: The Guilford Press.

Cohen, J. (1998) "Statistical Power Analysis for the Behavioral Sciences", NY: Routledge Academic.

Comrey, A. L. Lee, H. B. (1992) "A first course in factor analysis", 2ª ed. Hillsdale NJ: Lawrence: Erlbaum Associates.

Cronbach, L. J. (1951), "Coefficient alpha and the internal structure of tests", Psychometrika, v. 16, n. 3, p. 297–334.

CSTA (2016/2017) "Computer Science Framework/Standards", CSTA.

DeVellis, R. F. (2003) "Scale development: theory and applications", Thousand Oaks: SAGE Publications.

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

Golin, E. J., Reiss, S. P. (1990) "The Specification of Visual Language Syntax", Journal of Visual Languages and Computing, v. 1, n. 2, p. 141-157.

Gresse von Wangenheim, C. et al. (2018) "CodeMaster - Automatic Assessment and Grading of App Inventor and Snap! Programs". Informatics in Education, v. 17, n. 1, p. 117-150.

Grover, S., Pea, R. (2013) "Computational Thinking in K–12 A review of the state of the field", Educational Researcher, v. 42, n. 1, p. 38-43.

Grover, S. et al. (2014) "Assessing computational learning in K-12", Proc. of the Conf. on Innovation & Technology in Comp. Science Education, Uppsala, Suécia, p. 57–62.

IBGE (2015) "Pesquisa Nacional por Amostra de Domicílios - Posse de telefone móvel celular para uso pessoal", Instituto Brasileiro de Geografia e Estatística, Brasil.

Jackson, J. E. (2005) "Oblimin Rotation". Encyc. of Biostatistics, John Wiley&Sons, Ltd.

Larman, C., Basili, V. (2003) "Iterative and incremental developments: a brief history". IEEE Computer, v. 36, p. 47–56.

McCauley, R. (2003) "Rubrics as Assessment Guides". Newsletter ACM SIGCSE Bulletin, vol. 35, no. 4, pp. 17-18.

MEC (2018) "Base Nacional Comum Curricular", Brasil.

Moreno-León, J.; Robles, G. (2015) "Dr. Scratch: A Web Tool to Automatically Evaluate Scratch Projects". Proc. of the Workshop in Primary and Secondary Computing Education, Londres, Reino Unido.

Ota, G.; Morimoto, Y.; Kato, H. (2016) "Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts". Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing, Cambridge, Reino Unido, p. 238-239.

Petersen, K. et al. (2008) "Systematic mapping studies in software engineering". Proc. of the Int. Conf. on Evaluation and Assessment in Soft. Eng., Swindon, UK, p. 68-77.

Santos, P. S. C.; Araujo, L. G. J.; Bittencourt, R. A. (2018) "A Mapping Study of Computational Thinking and Programming in Brazilian K-12 Education". Proc. of the 48th Annual Frontiers in Education Conference, San Jose, CA.

Sherman, M.; Martin, F. (2015) "The assessment of mobile computational thinking", Journal of Computing Sciences in Colleges, v. 30, n. 6, p. 53–59.

Torrance, H. (1995) "Evaluating authentic assessment: Problems and possibilities in new approaches to assessment", Buckingham: Open University Press.

Turbak, F. et al. (2014) "Events-first programming in APP inventor", Journal of Computing Sciences in Colleges, v. 29, n. 6, p. 81- 89.

Wing, J. M. (2006) "Computational thinking". Comm. of the ACM, v. 49, n. 3, p. 33–35.

Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P., Clayborn, L. (2015) "Sowing the Seeds: A Landscape Study on Assessment in Secondary Computer Science Education", Proc. of the CSTA Annual Conference, Grapevine, TX.

Yin, R. K. (2001) "Case study research: design and methods". Ed. 2, Thousand Oaks: SAGE Publications.