UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Igor da Silva Solecki

# Uma abordagem para a avaliação do design visual de aplicativos móveis criados com linguagens de programação baseadas em blocos

Florianópolis
2020

Igor da Silva Solecki

# Uma abordagem para a avaliação do design visual de aplicativos móveis criados com linguagens de programação baseadas em blocos

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do título de Mestre em Ciência da Computação.
Orientador: Prof.ª Christiane Gresse von Wangenheim, Dr. rer. nat., PMP

Florianópolis
2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Solecki, Igor da Silva
Uma abordagem para a avaliação do design visual de aplicativos móveis criados com linguagens de programação baseadas em blocos / Igor da Silva Solecki ; orientadora, Christiane Anneliese Gresse von Wangenheim, 2020.
99 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação, Florianópolis, 2020.

Inclui referências.

1. Ciência da Computação. 2. Avaliação Educacional. 3. Design de Interface de Usuário. I. Gresse von Wangenheim, Christiane Anneliese. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Igor da Silva Solecki

**Uma abordagem para a avaliação do design visual de aplicativos móveis criados com linguagens de programação baseadas em blocos**

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Ricardo Azambuja Silveira, Dr.
Universidade Federal de Santa Catarina

Prof. Cristian Cechinel, Dr.
Universidade Federal de Santa Catarina

Prof. Rafael de Santiago, Dr.
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Ciência da Computação.

_____

Coordenação do Programa de Pós-Graduação

_____

Profa. Christiane Anneliese Gresse von Wangenheim, Dra.
Orientadora

Florianópolis, 2020.

## AGRADECIMENTOS

Agradeço a Deus, que me deu tranquilidade mesmo quando as coisas pareciam estar dando errado no decorrer deste trabalho.

Agradeço à professora Christiane, cuja orientação me levou a aprender muito e a alcançar os resultados que alcancei, e tornou a realização deste trabalho uma experiência valiosa.

Agradeço aos meus colegas do Grupo de Qualidade de Software – GQS/UFSC e aos professores Jean e Adriano pelas suas contribuições indispensáveis em diversas etapas deste trabalho.

Agradeço aos professores que participaram como membros da banca pelos seus comentários e sugestões enriquecedores.

Agradeço à milha família, que sempre me incentivou e me deu o suporte necessário para concluir este trabalho.

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil.

## RESUMO

Competências relacionadas à computação são necessárias para atuar na sociedade, o que tem motivado o ensino de computação na educação básica. A computação frequentemente é ensinada por meio de atividades abertas de programação de aplicativos móveis com linguagens visuais. Nesse contexto, avalia-se a aprendizagem com base no artefato de código criado pelo aluno, identificando se os conceitos abordados foram aplicados corretamente. Embora já existam diversas abordagens para a avaliação de código de linguagens visuais, não se encontraram abordagens incluindo a avaliação do design de interface do usuário (IU) de aplicativos móveis de forma detalhada. Assim, o objetivo deste trabalho é desenvolver uma abordagem para a avaliação de design visual de aplicativos móveis desenvolvidos com linguagens de programação baseadas em blocos. A abordagem é instanciada por uma rubrica para a avaliação do design de IU de aplicativos criados com o ambiente App Inventor e automatizada evoluindo a ferramenta web CodeMaster. A abordagem é avaliada estatisticamente em relação à sua confiabilidade e validade utilizando aplicativos desenvolvidos com o App Inventor. Os resultados da avaliação indicam que a abordagem é confiável e válida. Disponibilizando essa abordagem, espera-se facilitar a avaliação do design de IU de aplicativos móveis criados com App Inventor, fornecendo suporte ao ensino desses conceitos na educação básica.



**Palavras-chave:** Avaliação educacional. Design de interface de usuário. Linguagem de programação baseada em blocos. Educação Básica.

## ABSTRACT

Competences related to computing are necessary to actively participate in society, which has motivated teaching computing in K–12. Computing is often taught through open-ended activities involving for example programming mobile apps with visual languages. In this context, learning is assessed based on the code artifact created by the student, identifying whether the concepts taught were applied correctly. Although there are already several approaches for assessing code created with visual languages, no approaches addressing the assessment of the user interface (UI) design of mobile apps in detail were found. Thus, the objective of this work is to develop an approach for the assessment of the visual design of mobile apps developed with blocks-based programming languages. The approach is instantiated by a rubric for the assessment of the UI design of apps created with the App Inventor environment and automated evolving the CodeMaster web tool. The approach is evaluated statistically concerning its reliability and validity using apps developed with App Inventor. The results of the evaluation indicate that the approach is reliable and valid. By making this approach available, it is expected to facilitate the assessment of the UI design of apps created with App Inventor, providing support for teaching these concepts in K–12.



**Keywords:** Educational assessment. User interface design. Blocks-based programming language. K–12.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| BNCC | Base Nacional Comum Curricular |
| CSTA | *Computer Science Teachers Association* |
| GQM | *Goal Question Metric* |
| NA | Não se Aplica |
| PPGCC | Programa de Pós-Graduação em Ciência da Computação |
| IU | Interface de Usuário |
| TCT | Teoria Clássica dos Testes |
| TRI | Teoria da Resposta ao Item |
| WCAG | *Web Content Accessibility Guidelines* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Este capítulo contextualiza a pesquisa, apresentando também o objetivo deste trabalho, o método de pesquisa utilizado para realizá-lo e as contribuições esperadas com a sua realização.

## 1.1 CONTEXTUALIZAÇÃO

Sistemas de software hoje são parte fundamental de muitas atividades do dia-a-dia (SALIM; HAQUE, 2015), o que constantemente motiva estudos sobre a qualidade na interação com esses sistemas (SALOMÓN et al., 2019). Acompanhando a produção crescente de *smartphones* (CAPELAS, 2017), houve também um aumento exponencial no desenvolvimento de aplicativos para dispositivos móveis na última década (WASSERMAN, 2010). A complexidade desses aplicativos cresceu ao longo do tempo, passando de entretenimento e utilidades gerais para aplicações voltadas a negócios, que exigem maior qualidade (NAGAPPAN; SHIHAB, 2016). Assim, tornou-se necessário estudar especificamente a qualidade de aplicativos móveis, visto que o seu desenvolvimento envolve aspectos em geral menos relevantes para outros tipos de software (WASSERMAN, 2010).

A qualidade de software depende de vários atributos, como usabilidade, desempenho, corretude, entre outros. Como parte do processo de desenvolvimento de software, diversas técnicas são utilizadas com o fim de avaliar e melhorar a qualidade considerando esses atributos (BOURQUE; FAIRLEY, 2014). Técnicas de análise estática consistem em examinar o código sem executá-lo para identificar problemas como possíveis *bugs* e complexidade desnecessária. Já a análise dinâmica consiste em executar o programa com certos conjuntos de dados, com o objetivo de descobrir erros e ajudar a avaliar a corretude (TRUONG; ROE; BANCROFT, 2004). Ferramentas que aplicam essas técnicas para avaliar a qualidade são utilizadas na indústria (BELLER et al., 2016), mas seu uso se estende para o contexto do ensino de computação, suportando a avaliação de código criado em atividades educacionais (IHANTOLA et al., 2010).

Atualmente, há iniciativas no mundo todo para o ensino de computação na educação básica, com a visão de que, no cenário atual, todos deveriam possuir uma compreensão clara dos princípios e práticas da computação, independentemente da área da carreira profissional (CSTA, 2016). A computação tipicamente é ensinada por meio de unidades instrucionais de programação, nas quais avalia-se a aprendizagem dos alunos por meio da avaliação de artefatos

criados por eles. A avaliação tradicionalmente é feita manualmente pelo professor, porém essa forma de avaliação torna-se inviável devido a diversos fatores, como o grande número de exercícios a corrigir, cansaço, favoritismo, entre outros (STANKOV et al., 2013) (FONTE et al., 2013). Para superar essas dificuldades inerentes à avaliação manual, uma solução amplamente difundida é o uso de avaliação automatizada (ALVES; GRESSE VON WANGENHEIM; HAUCK, 2019).

Técnicas de análise estática de código, incluindo as utilizadas no contexto educacional, costumam ter a finalidade de analisar código desenvolvido com linguagens de programação baseadas em texto, como Java ou C++ (IHANTOLA et al., 2010). No entanto, essas técnicas podem ser utilizadas também para avaliar código de linguagens de programação baseadas em blocos, que se popularizaram nos últimos anos (BAU et al, 2017), principalmente no contexto do ensino de computação na educação básica (GROVER; BASU; SCHANK, 2018). Linguagens de programação baseadas em blocos possibilitam a criação de programas funcionais utilizando um ambiente visual, em que os comandos e expressões válidos da linguagem são listados como blocos em vez de serem memorizados, e o código é desenvolvido encaixando esses blocos entre si, não sendo possível encaixá-los de forma sintaticamente inválida. Assim, essas linguagens abstraem a sintaxe, facilitando o aprendizado (LYE; KOH, 2014).

Especificamente para o desenvolvimento de aplicativos móveis no contexto da educação básica, geralmente adota-se o ambiente de programação baseado em blocos App Inventor. O App Inventor se tornou popular para o ensino de computação na educação básica devido à facilidade com que ele permite criar aplicativos para *smartphones* e à possibilidade de desenvolver aplicativos para resolver problemas do mundo real (GROVER; BASU; SCHANK, 2018) (PATTON; TISSENBAUM; HARUNANI, 2019) (WOLBER, 2011). Com o App Inventor, é possível criar aplicativos para Android, e está em andamento o suporte a iOS. O App Inventor está disponível *online* e pode ser acessado e utilizado gratuitamente. O número de usuários que utilizam o App Inventor todo mês atualmente passa de 1 milhão em 195 países (MIT, 2019).

Existem atualmente diversos cursos, tutoriais de programação e unidades instrucionais para ensinar a criação de aplicativos com o App Inventor (PATTON; TISSENBAUM; HARUNANI, 2019). Essas unidades instrucionais focam tipicamente apenas no ensino de conceitos de programação, como condicionais, operadores e eventos. No entanto, o desenvolvimento de aplicativos envolve não somente competências de programação, mas

também de design de interface, que integra a Interação Humano-Computador, um dos conceitos transversais no ensino de computação (CSTA, 2016). O design de interface deve maximizar a usabilidade e a experiência do usuário (NIELSEN, 1994), sendo que a usabilidade é um fator crítico para o sucesso de aplicativos móveis (LEE et al., 2015).

Uma parte importante do design de interface de usuário (IU) é o design visual, que considera os efeitos dos elementos visuais na usabilidade e na atratividade estética dos produtos para dar forma à experiência de usuário e melhorá-la. Para isso, o design visual baseia-se em meta-princípios, incluindo consistência, hierarquia e personalidade (SCHLATTER; LEVINSON, 2013). O design visual pode ser orientado por guias de estilo, como o Material Design (GOOGLE, 2019), que fornece recomendações para o design de interface de aplicativos Android, abordando elementos como cores, tipografia, *layout* e componentes. Além disso, diretrizes de acessibilidade como as *Web Content Accessibility Guidelines* 2.1 (WCAG 2.1; W3C, 2018) podem orientar o design visual, assegurando a acessibilidade do conteúdo para o maior número possível de usuários.

Tendo em vista a importância do design visual de aplicativos móveis, é necessário que se abordem também competências de design visual como parte do ensino de computação na educação básica (FERREIRA et al., 2019). Nesse contexto, é essencial acompanhar a aprendizagem dos alunos, avaliando o design visual de interface dos aplicativos criados por eles e fornecendo *feedback*.

No contexto de aprendizagem baseada em problemas, que envolve atividades complexas e "mal-estruturadas" (sem uma solução predefinida) de desenvolvimento de aplicativos móveis, a aprendizagem pode ser avaliada por meio de avaliações baseadas em desempenho, que são tipicamente suportadas pelo uso de rubricas. Rubricas consistem em um conjunto de critérios de avaliação, e, para cada critério, é definida uma escala de níveis de desempenho que varia de "baixo desempenho" ao desempenho desejado (ALLEN; TANNER, 2006). Assim, rubricas permitem avaliar se o desempenho, o processo e o progresso dos alunos estão de acordo com o esperado com base em artefatos produzidos por eles (WHITTAKER; SALEND; DUHANEY, 2001) (MCCAULEY, 2003). Para inferir se o aluno desenvolveu competências relacionadas à programação, tipicamente utilizam-se rubricas para avaliar um artefato de software desenvolvido por ele, atribuindo um nível de desempenho (pontuação) conforme as habilidades demonstradas (SRIKANT; AGGARWAL, 2013). As notas nas atividades são determinadas convertendo a pontuação da rubrica em notas.

Embora existam abordagens para avaliar código desenvolvido com linguagens de programação baseadas em blocos, ainda existem poucas ferramentas para avaliar aplicativos

móveis criados com o App Inventor (ALVES; GRESSE VON WANGENHEIM; HAUCK, 2019). As abordagens existentes focam principalmente em conceitos de pensamento computacional envolvidos em programação, como iteração e recursão, abordando aspectos de design de IU apenas de forma superficial. Por exemplo, a rubrica de pensamento computacional móvel (SHERMAN; MARTIN, 2015) apenas contabiliza o número de telas de um aplicativo, de forma similar à rubrica do CodeMaster – App Inventor (GRESSE VON WANGENHEIM et al., 2018) (ALVES, 2019), que mede o número de componentes visuais e a presença de elementos de organização. Além disso, não existem ferramentas que suportem a avaliação automatizada do design de IU de aplicativos Android com base em diretrizes de guias de estilo ou de acessibilidade (PORTO; BARBOSA; GRESSE VON WANGENHEIM, 2018). Assim, observa-se a necessidade de suportar a avaliação automatizada baseada em desempenho de aplicativos criados com o App Inventor para facilitar o ensino de computação na Educação Básica.

## 1.2 OBJETIVO

O objetivo deste trabalho é desenvolver uma abordagem para automatizar a avaliação do design visual de aplicativos móveis desenvolvidos com linguagens de programação baseadas em blocos. A avaliação é realizada com base na análise estática do código criado com uma linguagem de programação baseada em blocos, avaliando o grau de conformidade do design visual com a teoria e guias de design visual, e é guiada por uma rubrica.

**Pergunta de pesquisa.** Como oferecer uma avaliação automatizada do design visual da interface de usuário de aplicativos Android criados com uma linguagem de programação baseada em blocos no contexto da Educação Básica de forma confiável e válida?

### 1.2.1 Objetivos específicos

Objetivo 1: Analisar o estado da arte em relação a abordagens e ferramentas de automatização de avaliação do design visual de aplicativos criados com linguagens de programação baseadas em blocos.

Objetivo 2: Desenvolver um modelo de avaliação (rubrica) para avaliar o grau de conformidade com diretrizes de design visual de aplicativos móveis.

Objetivo 3: Evoluir a ferramenta CodeMaster para automatizar a avaliação do design visual de aplicativos criados com um ambiente de programação baseado em blocos.

Objetivo 4: Avaliar a abordagem e ferramenta desenvolvidas em um estudo de caso.

### 1.2.2 Delimitação de escopo

A análise do design de interface se limita a aspectos do design visual, incluindo apenas critérios cuja verificação possa ser automatizada para projetos criados com o ambiente de programação App Inventor para a plataforma Android. Não são abordados outros aspectos envolvidos na criação de aplicativos móveis, como pensamento computacional ou criatividade. Também não são abordadas diretrizes de design visual específicas para outras plataformas.

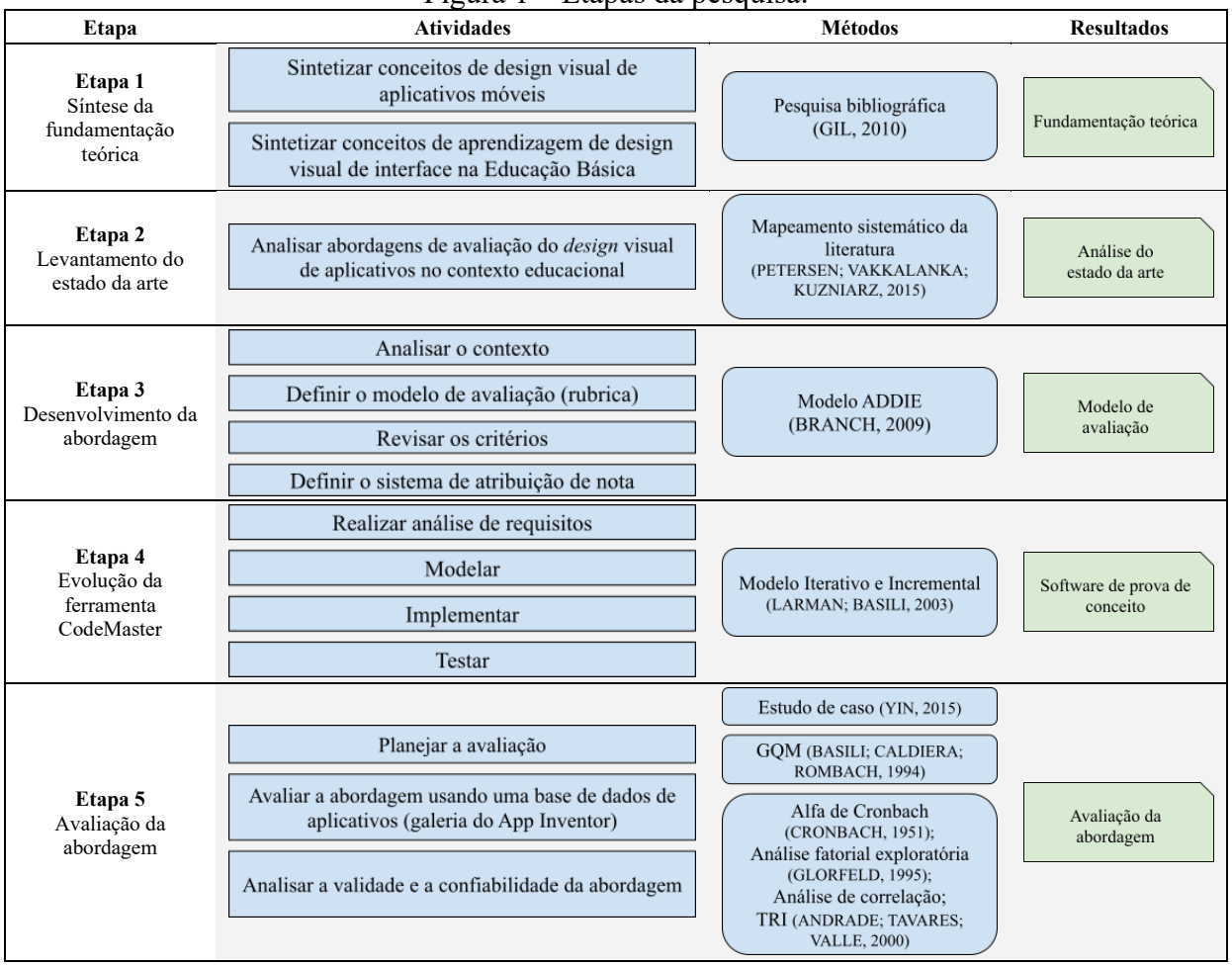
As técnicas de análise utilizadas na abordagem proposta se limitam à análise estática de código. Utiliza-se análise estática pois o design da interface é definido no código fonte, o que possibilita avaliar a aplicação de princípios de design visual analisando apenas o código. A análise dinâmica não é utilizada pois, em atividades abertas, que são o foco da abordagem desenvolvida, não há uma única solução correta ou resultados esperados que possam ser verificados por meio de análise dinâmica. A avaliação também não se baseia em *screenshots* do aplicativo em funcionamento, mas apenas na definição do design da interface registrada no código fonte.

Para a avaliação da abordagem, são utilizados aplicativos da galeria do App Inventor. Embora o contexto de desenvolvimento desses aplicativos não seja necessariamente o ensino de computação na Educação Básica, a galeria dá acesso a uma grande quantidade de aplicativos criados pela comunidade de usuários do App Inventor no mundo todo.

## 1.3 METODOLOGIA

Este trabalho caracteriza-se como uma pesquisa exploratória aplicada de abordagem multimétodos (SAUNDERS; LEWIS; THORNHILL, 2011): Exploratória, pois é direcionada conforme os resultados já obtidos, e de natureza aplicada, pois busca resolver questões específicas na análise automatizada de aplicativos criados com linguagens de programação baseadas em blocos. A pesquisa possui abordagem multimétodos, pois são utilizadas técnicas quantitativas e qualitativas, envolvendo pesquisa bibliográfica (GIL, 2010), mapeamento sistemático da literatura (PETERSEN; VAKKALANKA; KUZNIARZ, 2015), entre outros. As etapas da pesquisa são apresentadas na Figura 1.

Figura 1 – Etapas da pesquisa.




Fonte: Elaborada pelo autor

A seguir, cada etapa é descrita em detalhes.

**Síntese da fundamentação teórica.** O objetivo desta etapa é definir conceitos de design visual de aplicativos móveis e aprendizagem de design visual de interface como parte do ensino de computação na Educação Básica. Esta etapa é realizada sintetizando os conceitos relevantes para o trabalho por meio de uma pesquisa bibliográfica (GIL, 2010).

**Levantamento do estado da arte.** Nesta etapa, o estado da arte de abordagens de avaliação do design de interface de aplicativos Android é levantado por meio de um mapeamento sistemático da literatura (PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Na definição do protocolo, define-se a pergunta de pesquisa, os critérios de inclusão e exclusão dos resultados, a string de busca e as bases de dados em que a busca é realizada. Na execução, são realizadas buscas nas bases de dados definidas com a *string* de busca e os resultados são selecionados conforme os critérios de inclusão e exclusão. Na análise dos resultados, são

extraídas dos trabalhos selecionados e analisadas as informações relevantes para responder à pergunta de pesquisa.

**Desenvolvimento da abordagem.** A abordagem é desenvolvida utilizando o modelo ADDIE (BRANCH, 2009) de design instrucional. Inicialmente, caracteriza-se o público alvo e o ambiente por meio de uma análise de contexto, identificando também objetivos de aprendizagem e estratégias instrucionais para o ensino de design de IU no contexto de ensino de computação na educação básica. Com base no contexto identificado, define-se um modelo de avaliação baseado em uma rubrica definida segundo o procedimento proposto por Andrade (1997). De acordo com a teoria de design visual, definem-se critérios alinhados a diretrizes de guias de estilo e de acessibilidade para aplicativos móveis. Define-se um sistema de pontuação no qual os resultados da avaliação segundo a rubrica são convertidos em notas, de forma alinhada ao sistema de notas tipicamente adotado no Brasil.

**Desenvolvimento do suporte automatizado/evolução da ferramenta CodeMaster.** Utilizando um modelo iterativo e incremental de desenvolvimento de software (LARMAN; BASILI, 2003), a avaliação com base na rubrica é automatizada, evoluindo a ferramenta de software CodeMaster v1.0. Isso compreende iterativamente as fases de análise de requisitos, modelagem, implementação e testes.

**Avaliação da abordagem.** A abordagem desenvolvida é avaliada por um estudo de caso (YIN, 2015). Utilizando o método GQM (BASILI; CALDIERA; ROMBACH, 1994), define-se o que e como será avaliado. Para avaliar a abordagem, são coletados aplicativos da galeria do App Inventor e analisados automaticamente com a ferramenta CodeMaster utilizando a rubrica de design desenvolvida. Com esses dados, é realizada uma análise estatística quanto à confiabilidade da abordagem por meio do coeficiente Alfa de Cronbach (CRONBACH, 1951) e da sua validade por meio de uma análise da correlação do cumprimento de diretrizes com o grau de estética e a usabilidade. É realizada também uma análise fatorial exploratória (GLORFELD, 1995) para analisar a composição do modelo, e utiliza-se a Teoria da Resposta ao Item (TRI) (ANDRADE; TAVARES; VALLE, 2000) para analisar a validade das categorias dos itens. Os resultados são discutidos e interpretados, guiando o refinamento dos critérios e a identificação de novos critérios relevantes.

## 1.4 CONTRIBUIÇÕES

Este trabalho está inserido na linha de pesquisa de Engenharia de Software do PPGCC/INE/UFSC, especificamente ligado aos temas Computação na Escola e Interação

Humano-Computador (PPGCC, 2019). Esta seção apresenta as contribuições esperadas deste trabalho nos âmbitos científico, tecnológico e social.

**Contribuição científica.** A principal contribuição científica é um modelo de avaliação de código de uma linguagem de programação baseada em blocos referente à qualidade do design visual. Esta abordagem trata questões de qualidade pouco abordadas por trabalhos relacionados. Especificamente, espera-se o desenvolvimento/adaptação de técnicas de análise estática para a avaliação da estética e usabilidade do design de aplicativos móveis. A análise do estado da arte indica que, atualmente, não existem rubricas voltadas à avaliação do design visual de aplicativos criados com App Inventor no contexto da Educação Básica.

**Contribuição tecnológica.** A contribuição tecnológica resultante deste trabalho é um módulo de software funcional que auxilia instrutores em avaliar atividades de programação realizadas pelos alunos e auxilia os alunos em avaliar seus próprios projetos. Esse software evolui a ferramenta *web* CodeMaster v1.0, de modo a realizar uma avaliação mais completa de aplicativos criados com o App Inventor, focando na análise do design visual. A ferramenta desenvolvida está disponível online.

**Contribuição social.** Espera-se que o presente trabalho facilite o ensino da computação de forma ampla na educação básica. Automatizar a avaliação da aprendizagem de competências de design visual dará suporte ao ensino desse conteúdo, inclusive por instrutores sem formação na área de design de IU, visto que o *feedback* fornecido por meio da avaliação é essencial no processo de ensino. Espera-se também diminuir o esforço de avaliação e resolver outros problemas, como favoritismo etc. O contato com a computação na Educação Básica, especialmente com conceitos de design visual de interface, pode motivar alunos a se interessarem pela área. Além disso, a inclusão de conceitos e práticas da computação na educação básica favorece a inclusão digital, permitindo que uma maior parte da população tenha oportunidade de aprendê-los, proporcionando uma maior preparação para uma diversidade de carreiras.

## 1.5 ESTRUTURA DO DOCUMENTO

O restante deste documento está organizado da seguinte forma: No Capítulo 2, apresenta-se a fundamentação teórica, descrevendo os conceitos necessários para a compreensão do restante do trabalho. No Capítulo 3, um mapeamento sistemático das

abordagens existentes para a avaliação da interface de usuário de aplicativos Android é apresentado.

O Capítulo 4 apresenta o desenvolvimento da rubrica de avaliação do design visual de aplicativos criados com o App Inventor, CodeMaster UI design – App Inventor, e como a avaliação baseada na rubrica foi automatizada. O Capítulo 5 apresenta uma avaliação da rubrica utilizando aplicativos da galeria do App Inventor. Por fim, no Capítulo 6, a conclusão deste trabalho e as considerações finais são apresentadas.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados conceitos fundamentais de design visual com foco em aplicativos móveis, bem como diretrizes de usabilidade e acessibilidade para aplicativos Android. Este capítulo aborda também o ensino de design visual como parte do ensino de computação na educação básica e a avaliação automatizada da aprendizagem.

## 2.1 DESIGN VISUAL DE APLICATIVOS MÓVEIS

Como etapa final do design de IU, o design visual define em detalhes a aparência do aplicativo, tornando o design da interface concreto e determinando como a informação na tela será apresentada visualmente (GARRET, 2011). O design visual deve ser realizado levando em conta a estética visual, um fator importante para o sucesso de aplicativos móveis (XU; PEAK; PRYBUTOK, 2015) que afeta atributos como a usabilidade percebida e efetiva, satisfação subjetiva, entre outros. A estética dos elementos da interface pode aumentar o número de usuários interessados em utilizar o aplicativo e as suas chances de sucesso (WANG; LI, 2017).

O design de interface de aplicativos para dispositivos móveis se diferencia do design para outras plataformas devido ao espaço limitado da tela, à forma de interação com esses dispositivos e ao contexto em que são eles usados. Essas características devem ser consideradas ao realizar o design de interface para que ele seja efetivo (WASSERMAN, 2010).

O design visual envolve o uso de elementos como layout, cor, tipografia e imagens, e é guiado por meta-princípios (GARRET, 2011) (SCHLATTER; LEVINSON, 2013):

- Consistência: usar convenções no posicionamento e tratamento dos componentes para facilitar o entendimento da interface
- Hierarquia: destacar os elementos mais importantes para que chamem mais atenção
- Personalidade: efeitos na percepção dos usuários sobre o aplicativo

O uso balanceado dos elementos de design, seguindo os meta-princípios, pode resultar em uma interface bonita e amigável (GARRET, 2011).

**Diretrizes de design de interface de usuário.** Para auxiliar na aplicação dos princípios de design visual, o design pode ser orientado por guias de estilo, que fornecem diretrizes para facilitar o design de interfaces com usabilidade e acessibilidade. Especificamente para aplicativos Android, o Material Design (GOOGLE, 2019) se destaca como um guia de

estilo popular, sintetizando princípios clássicos de design. O cumprimento dessas diretrizes faz com que haja consistência com o design de outros aplicativos da mesma plataforma, satisfazendo as expectativas dos usuários e facilitando o uso (WASSERMAN, 2010). O Material Design contém diretrizes sobre a aparência dos componentes da interface e como eles devem reagir a interações do usuário. Por exemplo, a aparência de um botão pode ser uma das apresentadas na Figura 2, e o tipo mais apropriado depende da ênfase que se quer dar à ação. A ação principal de uma tela pode ser representada por um botão preenchido, ao lado de um botão de texto para uma ação secundária.

Figura 2 – Tipos de botão.


Botão preenchido Botão contornado Botão de texto

Fonte: Adaptado de Google (2019)

O Material Design ainda fornece diretrizes específicas para diversos outros componentes e aborda aspectos como *layout*, cor, tipografia e iconografia. Complementando essas diretrizes, as WCAG 2.1 (W3C, 2018) apresentam recomendações para garantir a acessibilidade da interface, abordando também aspectos do uso de cores, escrita e tipografia.

A seguir, são descritos os elementos do design de IU. São apresentadas também diretrizes do Material Design e das WCAG 2.1 e princípios de design visual referentes a esses elementos.

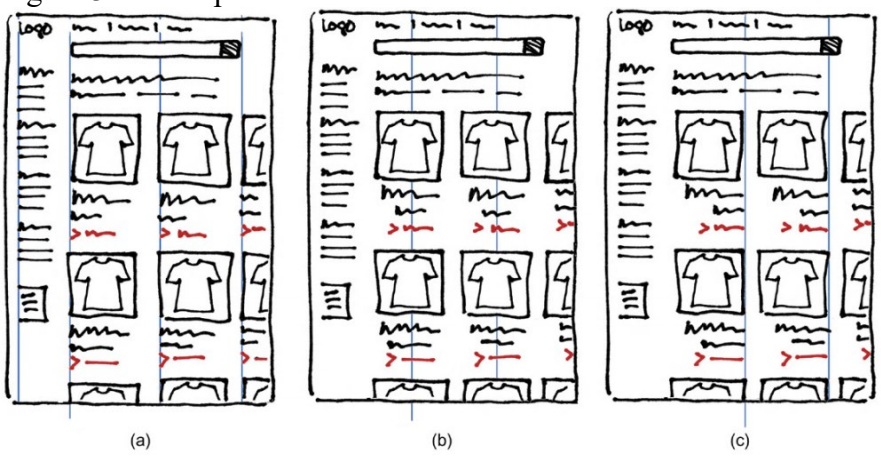
### 2.1.1 *Layout*

Definir o *layout* significa posicionar os componentes da interface seguindo uma certa estrutura, com o objetivo de ajudar os usuários a entenderem a interface. Entre os fatores envolvidos no *layout* estão o tamanho dos componentes e a proximidade entre eles, que afetam a percepção de quais componentes estão relacionados entre si e quais são mais importantes. O *layout* envolve também a forma como os componentes são alinhados (Figura 3).

O *layout* depende da resolução da tela (largura e altura), que determina o espaço disponível para posicionar os componentes. Especialmente em dispositivos móveis, que possuem espaço de tela limitado, deve-se evitar um excesso de informação em uma mesma tela,

focando nas informações essenciais, e apresentando detalhes em telas adicionais do aplicativo conforme necessário.

Figura 3 – Exemplo de uma mesma tela com diferentes alinhamentos




Alinhamento: (a) à esquerda, (b) centralizado ou (c) à direita. As linhas azuis verticais indicam a base do alinhamento.

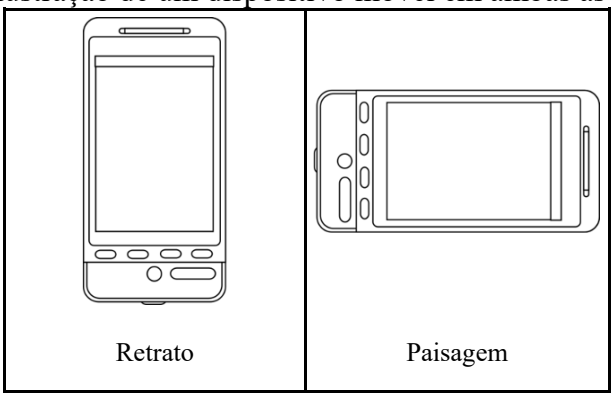
Fonte: Schlatter e Levinson (2013)

A resolução da tela é medida em pixels (px), mas o tamanho real de um pixel (o espaço físico que ele ocupa na tela do dispositivo) varia de dispositivo para dispositivo. Para compensar essas diferenças, a resolução pode ser medida em pixels independentes de densidade (dp), visto que 1dp ocupa aproximadamente o mesmo espaço físico em qualquer dispositivo. Assim, usar essa unidade de medida evita que os elementos da interface fiquem grandes ou pequenos demais dependendo do dispositivo. Para facilitar a interação, componentes alvos de toque, como botões, devem ter largura e altura de no mínimo 48dp.

A resolução da tela define a proporção da tela (*aspect ratio*, em inglês) e a orientação do dispositivo. A proporção da tela é a proporção entre a largura e a altura. Uma resolução de 480px × 640px, por exemplo, tem uma proporção de 3:4. A proporção de tela mais comum em dispositivos móveis atualmente é 9:16 (DEVICEATLAS, 2019).

A orientação da tela pode ser retrato (se altura > largura) ou paisagem (se altura < largura) (Figura 4). A maioria dos dispositivos móveis como *smartphones* e *tablets* utilizados atualmente podem ser usados em ambas as orientações (W3C, 2017). O que determina a orientação é a posição em que o usuário segura o dispositivo, exceto quando o aplicativo em uso força uma orientação específica.

Figura 4 – Ilustração de um dispositivo móvel em ambas as orientações.




Fonte: Wikipedia (2012)

### 2.1.2 Tipografia

A tipografia define o formato das letras e demais caracteres. Uma família tipográfica é composta por diversas fontes, que são variações de um mesmo formato básico. As variações mais comuns são itálico e negrito, mas também existem variações de outros tipos, como diversas espessuras de traço. Por exemplo, a família tipográfica Roboto, padrão do sistema Android, é formada pelas fontes apresentadas na Figura 5

Figura 5 – Fontes da família tipográfica Roboto.

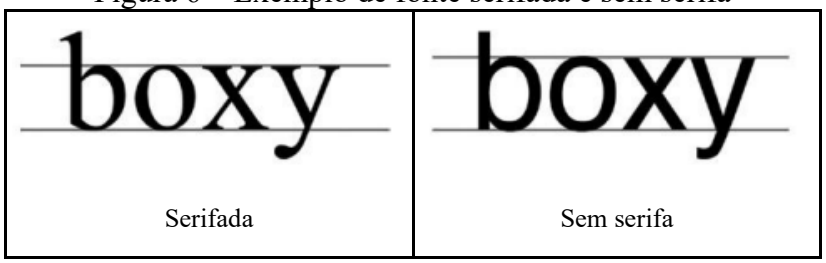

Fonte: Google (2018b)

A família tipográfica Roboto é do tipo sem serifa. Serifas são pontas que se estendem ao final dos traços. Os caracteres de fontes sem serifa possuem as pontas mais quadradas. Essa diferença é ilustrada na Figura 6.

Para que os usuários consigam identificar as funcionalidades do aplicativo com mais facilidade, deve-se usar uma quantidade limitada de estilos que indiquem claramente a importância relativa e a função de cada trecho de texto (cabeçalho, corpo do texto etc.). Para isso, o Material Design apresenta um conjunto de estilos predefinidos conforme o papel do texto na interface (Figura 7). Cada um desses estilos possui um tamanho de fonte definido, em geral

com tamanhos maiores para elementos mais destacados. Observa-se que todos os estilos utilizam a família tipográfica Roboto.

Figura 6 – Exemplo de fonte serifada e sem serifa




Fonte: Schlatter e Levinson (2013)

Figura 7 – Conjunto de estilos tipográficos do Material Design.

| Scale Category | Typeface | Font | Size | Case | Letter spacing |
|---|---|---|---|---|---|
| H1 | Roboto | Light | 96 | Sentence | -1.5 |
| H2 | Roboto | Light | 60 | Sentence | -0.5 |
| H3 | Roboto | Regular | 48 | Sentence | 0 |
| H4 | Roboto | Regular | 34 | Sentence | 0.25 |
| H5 | Roboto | Regular | 24 | Sentence | 0 |
| H6 | Roboto | Medium | 20 | Sentence | 0.15 |
| Subtitle 1 | Roboto | Regular | 16 | Sentence | 0.15 |
| Subtitle 2 | Roboto | Medium | 14 | Sentence | 0.1 |
| Body 1 | Roboto | Regular | 16 | Sentence | 0.5 |
| Body 2 | Roboto | Regular | 14 | Sentence | 0.25 |
| BUTTON | Roboto | Medium | 14 | All caps | 1.25 |
| Caption | Roboto | Regular | 12 | Sentence | 0.4 |
| OVERLINE | Roboto | Regular | 10 | All caps | 1.5 |

Fonte: Google (2019)

### 2.1.3 Escrita

A escrita é uma forma básica de comunicação, utilizada para apresentar conteúdo, fornecer instruções, identificar a função dos elementos, comunicar o propósito de uma tela ou

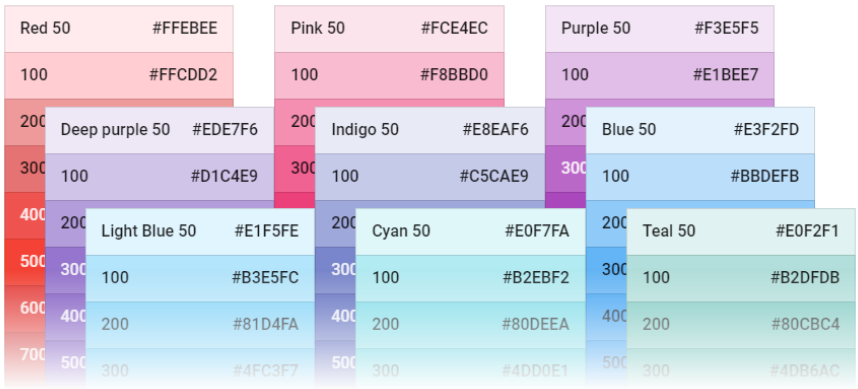
diálogo, informar o estado do aplicativo e dos processos que ele está executando, entre outros. A forma como o texto é redigido é crucial para seguir os meta-princípios.

O texto que compõe a interface em geral deve ser conciso e simples, comunicando os detalhes essenciais. Em particular, o texto de botões não deve ser excessivamente longo. Recomenda-se evitar pontuação quando esta não é essencial, omitindo dois pontos (:) ao final de legendas e ponto final (.) em frases isoladas. Para tornar o texto mais acessível, letras maiúsculas e minúsculas devem ser usadas de acordo com as regras de ortografia da língua em que o texto está escrito. Porém, no caso do texto de botões, o Material Design indica que todas as letras devem ser maiúsculas (ver Figura 7).

### 2.1.4 Cores

Na interface de um aplicativo, cores podem ser usadas para chamar a atenção para certos componentes, ajudar a identificar o que está relacionado e indicar o estado dos componentes. Assim, o uso de cores de forma apropriada pode melhorar a usabilidade, além de tornar a interface mais atrativa (SCHLATTER; LEVINSON, 2013).

Figura 8 – Parte da paleta de cores do Material Design.

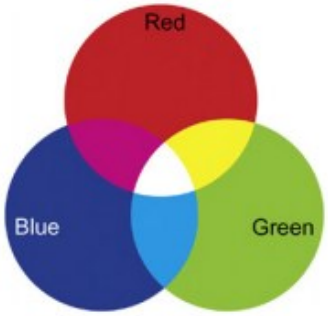



Fonte: Google (2019)

As paletas de cores de aplicativos móveis devem ser compostas basicamente por uma cor primária e uma cor secundária. As cores não devem ser aplicadas aleatoriamente, mas devem ser usadas para facilitar o entendimento da interface. Para facilitar a escolha de cores agradáveis para a interface, o Material Design disponibiliza uma paleta de cores predefinidas, composta de diversas tonalidades e suas variantes (Figura 8).

Um aspecto importante do esquema de cores utilizado na interface é se a combinação das cores é harmônica, ou seja, se as cores possuem uma relação de harmonia. As relações de harmonia são definidas com base no sistema de cores usado. O sistema de cores tradicional da pintura é o RYB (COLOR MATTERS, 2019a), que considera vermelho, amarelo e azul como cores primárias (do inglês ***R**ed*, ***Y**ellow*, ***B**lue*). Porém, o sistema usado para apresentar cores na tela de dispositivos digitais é o RGB (COLOR MATTERS, 2019b), em que as cores primárias são vermelho, verde e azul (do inglês ***R**ed*, ***G**reen*, ***B**lue*) (Figura 9).

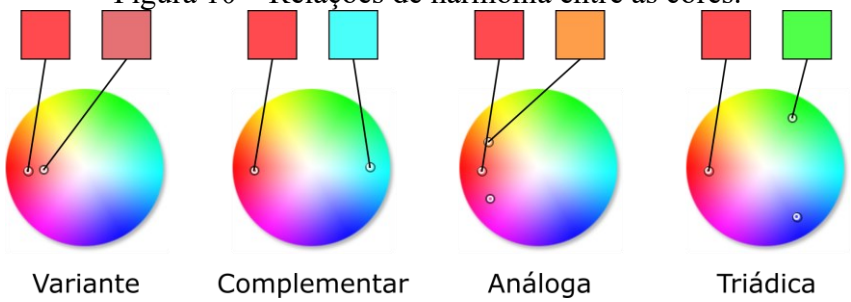
Figura 9 – Cores no sistema RGB.




Fonte: Schlatter e Levinson (2013)

Misturando as cores primárias, formam-se cores intermediárias. As cores primárias e cores intermediárias dispostas em círculo formam um círculo cromático. A posição das cores no círculo cromático define as relações de harmonia entre as cores. Algumas relações básicas de harmonia são apresentadas na Figura 10.

Figura 10 – Relações de harmonia entre as cores.

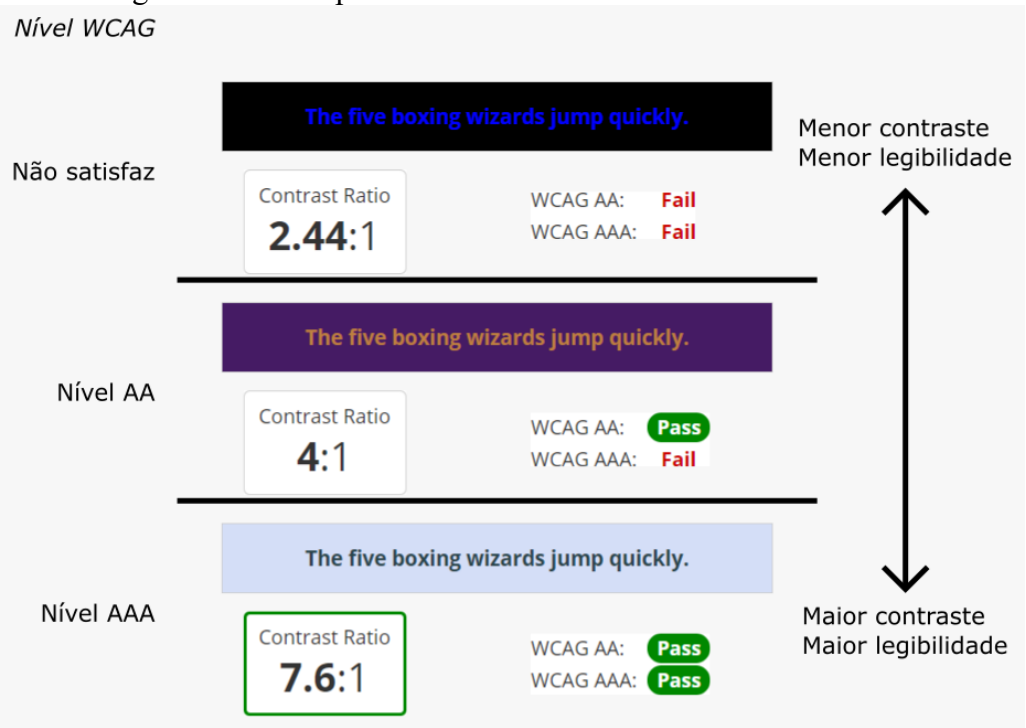



Fonte: Adaptado de Sessions College (2019)

Ao definir as cores do aplicativo, especialmente do texto, é necessário considerar o contraste com a cor de fundo. O menor contraste possível é expresso como 1:1, que não apresenta nenhuma legibilidade, e o maior contraste, 21:1. Para garantir a acessibilidade, as WCAG 2.1 estabelecem níveis mínimos de contraste, definindo dois níveis de conformidade:

nível AA e nível AAA. Um aplicativo satisfaz o nível AA se todo o texto possui contraste acima do mínimo, e o nível AAA requer um contraste maior que o nível AA (Figura 11).

Figura 11 – Exemplos de contraste da cor do texto com o fundo




Fonte: Adaptado de WebAIM (2019)

Conforme as WCAG 2.1, o contraste mínimo necessário difere entre texto considerado "pequeno" e texto "grande", o que é definido com base no tamanho da fonte e no uso do estilo negrito (Tabela 1). Texto "grande" é mais legível e, portanto, exige menos contraste (Tabela 2).

Tabela 1 – Definição de texto pequeno e grande conforme as WCAG 2.1.

| | **Não negrito** | **Negrito** |
|---|---|---|
| **Tamanho < 14pt** | Texto pequeno | |
| **14pt <= Tamanho < 18pt** | Texto pequeno | Texto grande |
| **Tamanho >= 18pt** | Texto grande | |

Fonte: W3C (2018)

Tabela 2 – Níveis mínimos de contraste exigidos pelas WCAG 2.1.

| | **Texto pequeno** | **Texto grande** |
|---|---|---|
| **Nível AA** | 4.5:1 | 3:1 |
| **Nível AAA** | 7:1 | 4.5:1 |

Fonte: W3C (2018)

### 2.1.5 Imagens

Diversos tipos de imagens podem ser usados na interface de um aplicativo, incluindo fotografias, vídeos, ilustrações, animações, logos, ícones etc. Assim, imagens têm diversos usos, podendo representar conteúdo, expressar uma marca ou estilo, apresentar detalhes ou exemplos, entre outros.

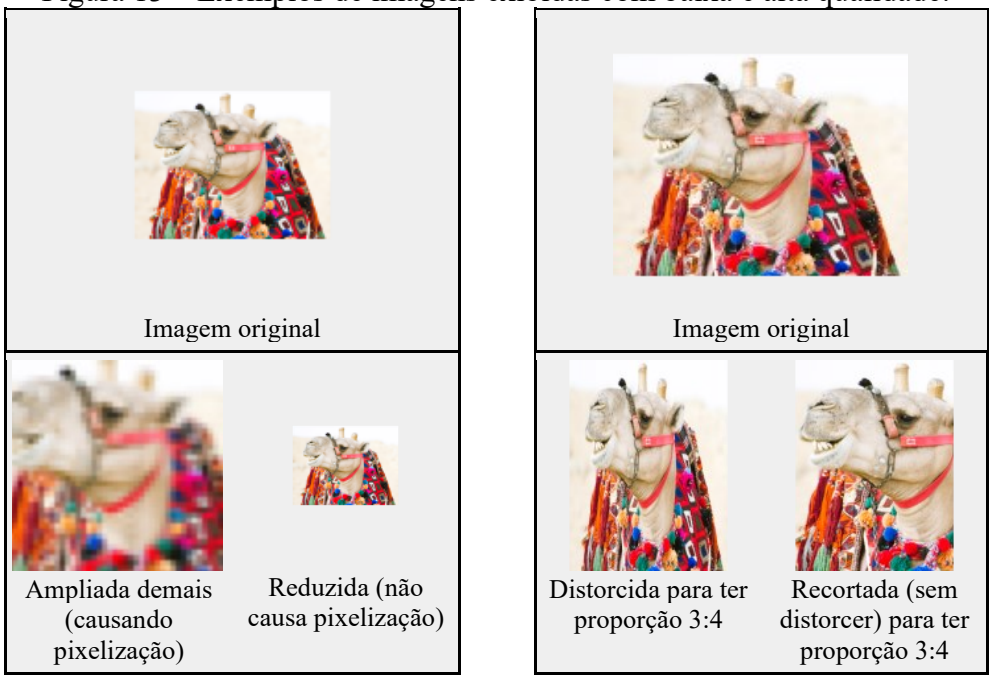
Ícones são um tipo específico de imagem utilizado para representar ações, objetos e conceitos na interface, tornando-os mais fáceis de identificar (LIDWELL; HOLDEN; BUTLER, 2010). O Material Design contém um catálogo de ícones prontos que podem ser usados gratuitamente na interface de qualquer aplicativo. Esse catálogo é composto de centenas de ícones com um estilo minimalista consistente, representando diversas ações e itens comuns de aplicativos móveis (Figura 12).

Figura 12 – Exemplos de ícones do catálogo do Material Design.


Fonte: Google (2019)

A qualidade das imagens é importante para transmitir mensagens de maneira efetiva. Assim, uma imagem não deve estar embaçada ou pixelizada, o que acontece quando ela é ampliada demais (Figura 13 a). Outro aspecto necessário para manter a qualidade da imagem é apresentá-la com a proporção correta. Para usar uma imagem com uma proporção diferente da original, ela deve ser cortada, e não distorcida (Figura 13 b).

Figura 13 – Exemplos de imagens exibidas com baixa e alta qualidade.



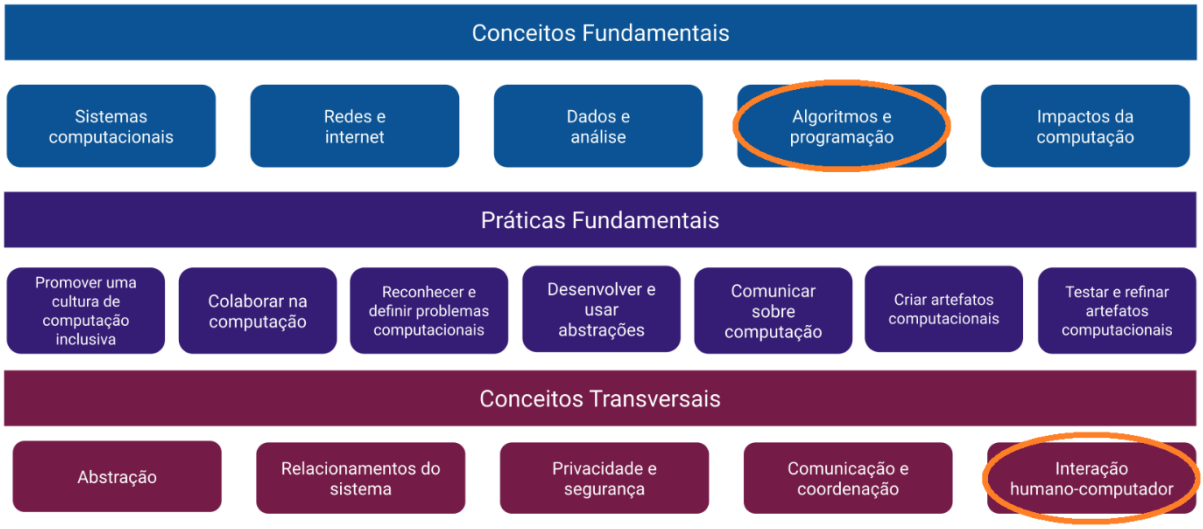

a) Pixelização b) Distorção

Fonte: Adaptado de Google (2019)

## 2.2 ENSINO DO DESIGN VISUAL COMO PARTE DO ENSINO DE COMPUTAÇÃO NA EDUCAÇÃO BÁSICA

O ensino de computação na educação básica envolve conceitos de pensamento computacional, processamento de informação e impactos da computação à sociedade (SBC, 2018). De forma mais detalhada, o *K–12 Computer Science Framework* (CSTA, 2016) define conceitos e práticas que devem guiar o ensino de computação na educação básica (Figura 14). As práticas do *framework* são atitudes necessárias para que os alunos estejam preparados para lidar com a tecnologia do mundo atual, enquanto os conceitos centrais representam áreas de conteúdo da ciência da computação. Complementando os conceitos centrais, os conceitos transversais são temas que estabelecem ligações entre os conceitos centrais. Segundo o *framework*, as práticas devem ser desenvolvidas ao longo da educação básica, e os conteúdos pertinentes a cada faixa etária são definidos com base nos conceitos. Os conceitos e práticas são abordados de forma mais específica pelo guia de currículo da *Computer Science Teachers Association* (CSTA) (CSTA, 2019).

Figura 14 – Práticas e conceitos do *framework* de computação para a Educação Básica (CSTA, 2016).




O ensino de computação tipicamente foca em Algoritmos e Programação. O design visual está relacionado à Interação Humano-Computador.

Fonte: CSTA (2016)

O foco de unidades instrucionais e cursos para o ensino de computação na educação básica tipicamente está no conceito de Algoritmos e Programação. No entanto, outro conceito importante é o conceito transversal Interação Humano-Computador. A Interação Humano-Computador abrange o design visual da interface, que é um fator de sucesso para aplicativos móveis, conforme apresentado na Seção 2.1. Assim, o design visual de interface está entre os conceitos que integram uma abordagem mais completa na aprendizagem de computação, conforme a visão do *framework*.

Uma forma de ensinar computação na educação básica que possibilita também o ensino de design visual é por meio do desenvolvimento de aplicativos móveis como parte de atividades abertas (que não possui apenas uma solução correta) (WOLBER; ABELSON; FRIEDMAN, 2015) (PATTON; TISSENBAUM; HARUNANI, 2019). Seguindo essa abordagem, os alunos desenvolvem aplicativos, por exemplo, para resolver um problema na sua comunidade, de forma que os aplicativos resultantes podem chegar a ter um impacto social positivo. Esse tipo de atividade está alinhado ao conceito de *ação computacional* (PATTON; TISSENBAUM; HARUNANI, 2019) (TISSENBAUM; SHELDON; ABELSON, 2019), uma visão para o aprendizado de computação de que os alunos devem desenvolver aplicativos focando em questões que causem impacto na vida deles, e seguindo um processo que se assemelha na medida do possível às práticas de desenvolvimento profissional, demonstrando assim a utilidade e importância das competências de computação.

Dada a importância do ensino de design visual como parte do ensino de computação, observa-se sua inclusão no guia de currículo para a Educação Básica do CSTA (2019), que define objetivos de aprendizagem alinhados com os conceitos e práticas do *framework*. Dentro desse contexto, os objetivos especificamente relacionados ao design visual são apresentados na Tabela 3.

Tabela 3 – Objetivos de aprendizagem do CSTA (2019) relacionados ao design visual.

| **Identificador** | **Objetivo** | **Descrição** |
|---|---|---|
| 2-CS-01 | Recomendar melhorias para o design de dispositivos de computação, com base em uma análise de como usuários interagem com os dispositivos | O estudo de interação Humano-Computador pode melhorar o design de hardware e software. Os alunos devem considerar a usabilidade de diversas perspectivas, incluindo a acessibilidade. |
| 2-IC-21 | Discutir questões de viés e acessibilidade no design de tecnologias existentes | Os alunos devem testar e discutir a usabilidade de aplicativos, dispositivos etc., notando, por exemplo, como alterações no design da interface podem facilitar o uso. |
| 3A-AP-21 | Avaliar e refinar artefatos computacionais para torná-los mais usáveis e acessíveis | Os alunos devem considerar as necessidades e expectativas dos usuários finais para melhorar a usabilidade e acessibilidade de artefatos, adequando, por exemplo, a interface de um aplicativo. |

Fonte: CSTA (2019)

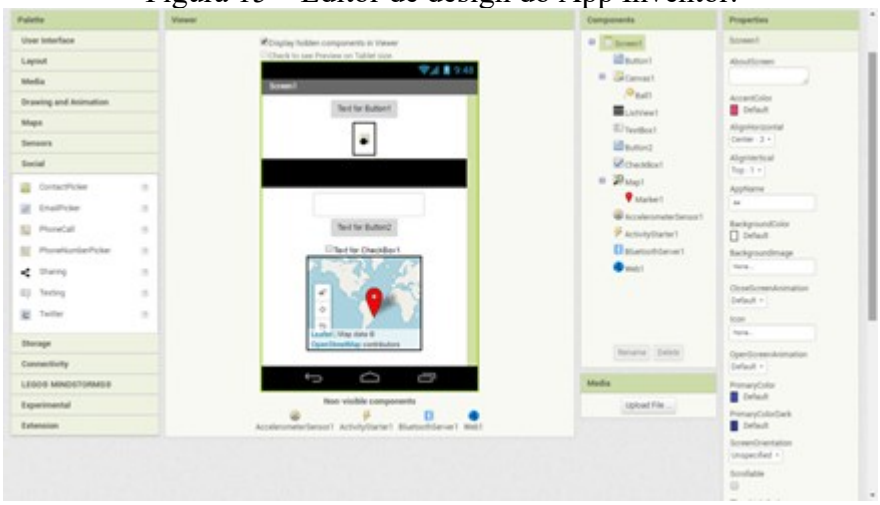
Na prática, existem diversas unidades instrucionais para o ensino de competências de design de IU no contexto da Educação Básica (FERREIRA et al., 2019a). Essas unidades instrucionais preveem a aplicação dos conceitos ensinados por meio de exercícios práticos, e frequentemente envolvem o desenvolvimento de software utilizando linguagens de programação baseadas em blocos. A avaliação da aprendizagem de design de interface tipicamente é realizada por meio de avaliações baseadas no desempenho, analisando artefatos criados pelos alunos em atividades da unidade instrucional. O ensino de design visual em específico envolve a escolha dos elementos (cores, ícones, tipografia e imagens), o que pode ser orientado por diretrizes de guias de estilo de plataforma (FERREIRA, 2019b).

### 2.2.1 App Inventor para o ensino de design visual

O App Inventor (MIT, 2019) é um ambiente de programação baseado em blocos para criar aplicativos móveis para dispositivos Android. Ele pode ser acessado e utilizado online gratuitamente por meio de um navegador *web*. O App Inventor consiste em um ambiente de programação visual em que se arrastam e posicionam blocos e componentes para desenvolver o aplicativo.

Os aplicativos são criados em dois passos no App Inventor. Inicialmente, componentes da IU, como botões e imagens, são posicionados e configurados no editor de design (Figura 15). A descrição dos componentes visuais do App Inventor é apresentada na Tabela 4.

Figura 15 – Editor de design do App Inventor.



Fonte: MIT (2019)

A aparência desses componentes pode ser configurada alterando propriedades como tamanho, cor, fonte, alinhamento, entre outros. Uma lista das propriedades que afetam a aparência dos componentes é apresentada no Apêndice A, relacionando quais componentes possuem cada propriedade.

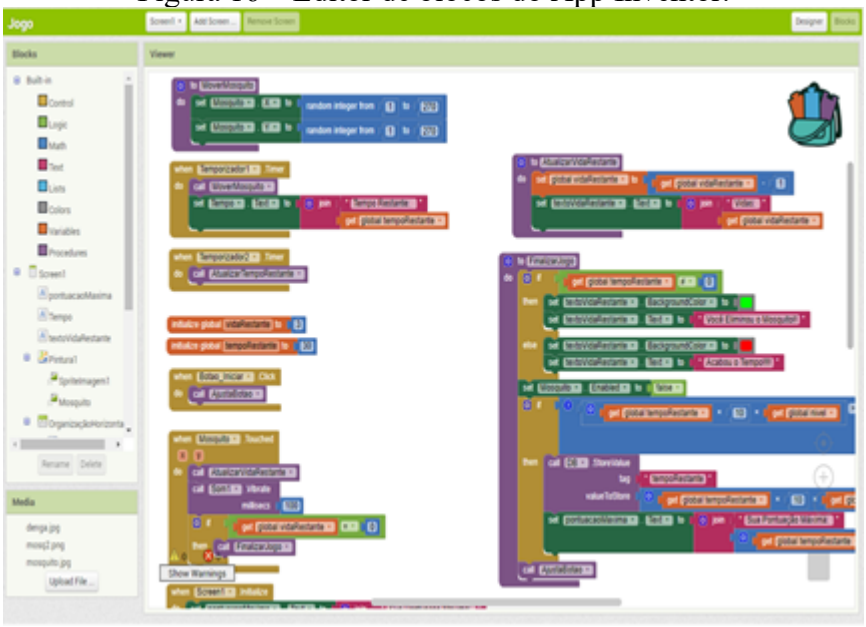
Num segundo passo, o funcionamento do aplicativo é programado conectando blocos visuais. Blocos de programação do tipo *built-in* representam conceitos tradicionais de programação (condicionais, laços, procedimentos etc.). Adicionalmente, existem blocos de programação para elementos do editor de design, que representam eventos, estados e ações de componentes específicos (p. ex., botão pressionado, ativar temporizador) (MUSTAFARAJ; TURBAK; SVANBERG, 2017). O funcionamento do aplicativo é definido no Editor de Blocos (Figura 16).

Tabela 4 – Componentes visuais do App Inventor.

| **Componente** | **Descrição** |
|---|---|
| Botão | Componente capaz de detectar cliques. Quando clicado, dispara alguma ação. |
| CaixaDeSeleção | Componente capaz de detectar toques do usuário, alterando seu estado booleano em resposta. |
| EscolheData | Um botão de propósito especial. Quando clicado, exibe um diálogo para o usuário escolher uma data. |
| Imagem | Componente para exibir imagens. |
| Legenda | Componente para mostrar texto. |
| EscolheLista | Um botão de propósito especial. Quando clicado, exibe uma lista de textos e pede que o usuário escolha um elemento da lista. |
| VisualizadorDeListas | Exibe uma lista de textos como parte da tela. |
| Notificador | Exibe diálogos de alerta, mensagens e alertas temporários. |
| CaixaDeSenha | Uma caixa de texto que esconde o texto digitado nela. |
| Tela | Componente que contém todos os outros componentes. |
| Deslizador | Uma barra com um pino arrastável. À medida que o usuário arrasta o pino para os lados, a posição é notificada. |
| ListaSuspensa | Componente que exibe um *popup* com uma lista de elementos para o usuário selecionar um item. |
| Switch | Detecta toques do usuário e muda seu estado binário em resposta. |
| CaixaDeTexto | Componente que permite que o usuário insira texto. |
| EscolheHora | Um botão de propósito especial. Quando clicado, exibe um diálogo para o usuário escolher uma hora. |
| NavegadorWeb | Componente para visualizar páginas *web*. |
| EscolheImagem | Um botão de propósito especial. Quando clicado, mostra a galeria de imagens do dispositivo para que o usuário escolha uma imagem. |
| ReprodutorDeVídeo | Reproduz vídeos em uma área retangular na interface. |
| Pintura | Painel retangular no qual é possível fazer desenhos ou mostrar sprites. |
| Mapa | Apresenta um mapa embutido na interface. |
| EscolheContato | Um botão de propósito especial. Quando clicado, mostra uma lista de contatos para que o usuário escolha um deles. |
| EscolheEmail | Uma caixa de texto que sugere contatos quando o usuário começa a digitar um nome ou e-mail. |
| EscolheNúmeroDeTelefone | Um botão de propósito especial. Quando clicado, mostra uma lista dos números de telefone dos contatos para que o usuário escolha um deles. |
| | *Componentes de organização* |
| OrganizaçãoHorizontal | Dispõe outros componentes linearmente, da esquerda para a direita. |
| HorizontalScrollArrangement | O mesmo que OrganizaçãoHorizontal, com a função adicional de rolamento. |
| OrganizaçãoEmTabela | Dispõe outros componentes de forma tabular. |
| OrganizaçãoVertical | Dispõe outros componentes linearmente, de cima para baixo. |
| VerticalScrollArrangement | O mesmo que OrganizaçãoVertical, com a função adicional de rolamento. |

Fonte: MIT (2019)

Figura 16 – Editor de blocos do App Inventor.



Fonte: MIT (2019)

O aplicativo pode ser testado à medida que é desenvolvido por meio do aplicativo App Inventor Companion. O App Inventor Companion executa o aplicativo em desenvolvimento em tempo real em um dispositivo móvel. Assim que o aplicativo estiver pronto, é possível gerar um arquivo executável .apk, que pode ser publicado na Play Store (loja *online* da Google) ou distribuído de outras formas.

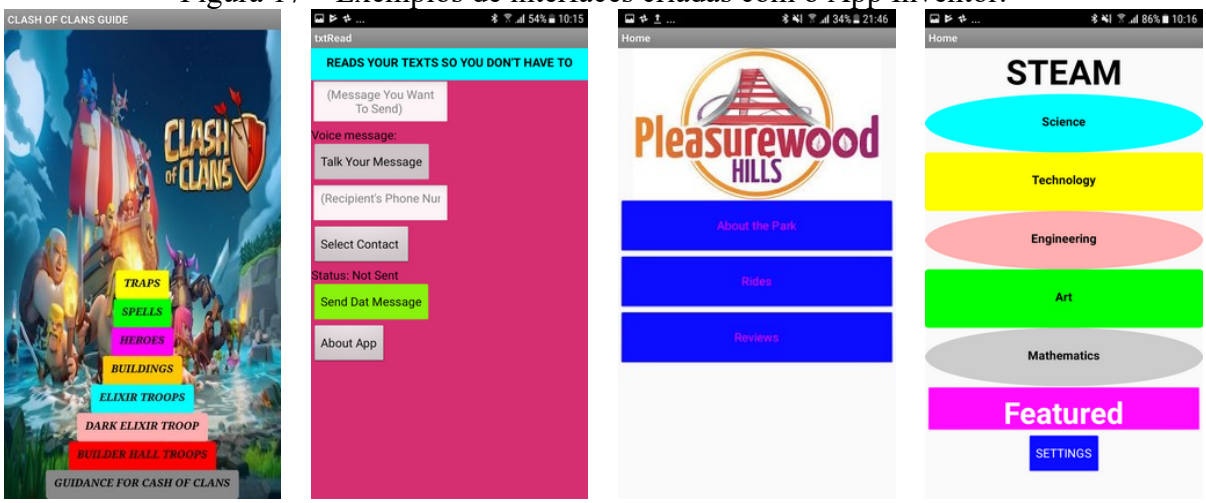
Os arquivos de código fonte de projetos do App Inventor são salvos automaticamente na nuvem, mas podem também ser exportados como arquivos “.aia”. Um arquivo “.aia” é um conjunto de arquivos compactados que inclui um arquivo de propriedades do projeto, arquivos de mídia usados pelo aplicativo e, para cada tela do aplicativo, um arquivo “.bky” e um arquivo “.scm”. O arquivo “.bky” contém uma estrutura XML com os blocos de programação usados na lógica do aplicativo, e o arquivo “.scm” contém uma estrutura JSON descrevendo os componentes usados (MUSTAFARAJ; TURBAK; SVANBERG, 2017).

O App Inventor é um software de código aberto disponível no GitHub (https://github.com), de forma que qualquer pessoa pode baixá-lo e configurá-lo em um servidor próprio (WOLBER; ABELSON; FRIEDMAN, 2015).

**Aplicação dos princípios de design visual.** Uma vez que o App Inventor permite configurar uma variedade de aspectos da aparência dos componentes (conforme indicado no Apêndice A), é possível realizar o design de interface dos aplicativos seguindo os princípios e diretrizes de design visual. No entanto, ao observar a interface de aplicativos encontrados na

galeria do App Inventor (Figura 17), nota-se que o design visual das interfaces frequentemente não está alinhado aos princípios e diretrizes.

Figura 17 – Exemplos de interfaces criadas com o App Inventor.




a) b) c) d)

Fonte: Elaborada pelo autor

Conforme exemplificado na Figura 17, as interfaces apresentam problemas como:

a) Imagens distorcidas (imagem de fundo);
b) Cores não harmônicas (cores sem relação harmônica);
c) Contraste insuficiente (texto dos botões);
d) Falta de consistência (botões de cores e formatos arbitrários).

Observa-se que essas violações dos princípios e diretrizes são frequentes entre os aplicativos da galeria. Visto que a galeria contém aplicativos desenvolvidos por usuários do mundo todo, ela fornece uma visão geral do design visual de aplicativos criados pelos usuários do App Inventor (SOLECKI et al., 2020), indicando a necessidade do ensino de conceitos de design visual.

## 2.3 AVALIAÇÃO DO DESIGN VISUAL NO CONTEXTO EDUCACIONAL

A avaliação no contexto educacional visa a fornecer *feedback* ao aluno e/ou ao professor em relação à aprendizagem das competências pelo aluno (BRANCH, 2009). Assim, a avaliação é um elemento central no processo de aprendizagem.

A avaliação *formativa* tem o objetivo de auxiliar o aluno a desenvolver competências, adequando o processo de ensino com base no seu desempenho. A avaliação *somativa*, por outro

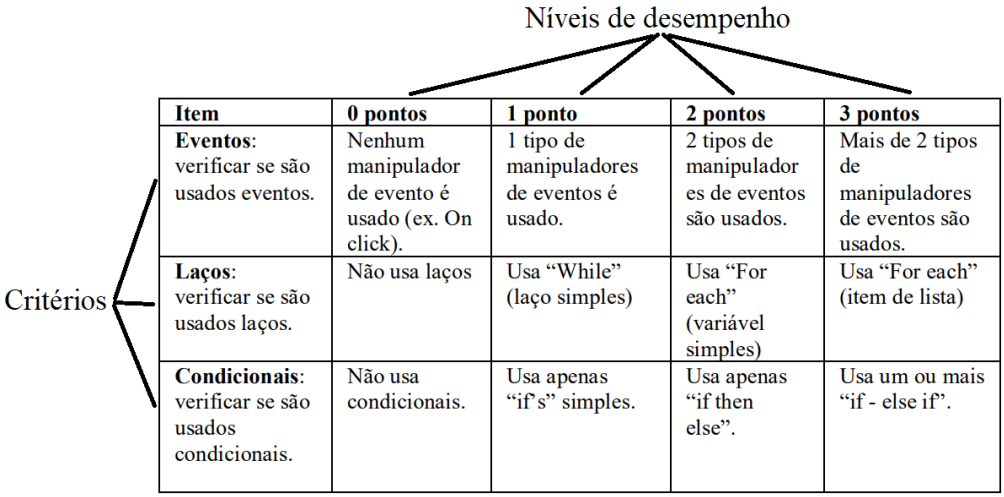
lado, fornece uma visão geral do que o aluno alcançou ao final de uma unidade instrucional, não tendo impacto direto no processo de aprendizagem (SADLER, 1989). Um elemento essencial da avaliação formativa é o *feedback*, que informa quão distante o aluno está do desempenho desejado. Uma característica importante do *feedback* é que ele serve para melhorar o desempenho do aluno, e não apenas indicar que o desempenho desejado não foi atingido. Assim, o *feedback* deve ser fornecido de forma que motive o aluno a se esforçar e possa ser usado para ajustar a aprendizagem (WILIAM, 2011).

Como resultado da avaliação, tipicamente se atribui uma nota conforme a qualidade dos resultados do aluno. No Brasil, não há um sistema de notas padronizado, mas é comum na Educação Básica a atribuição de uma nota numérica de 0 (desempenho totalmente insatisfatório) a 10 (desempenho totalmente satisfatório) (CME, 2011). Esse sistema é utilizado na avaliação de diversas atividades desempenhadas pelos alunos. No entanto, para fins de avaliação formativa, apresentar apenas uma nota sem tornar explícitos os julgamentos e critérios usados para determiná-la não é útil como *feedback* (SADLER, 1989).

A avaliação pode ser feita de diversas formas, como provas, autoavaliação dos alunos, relatórios orais ou escritos, e por desempenho. (IRVINE; CROWLEY, 2013) A avaliação baseada no desempenho avalia as competências dos alunos com base em artefatos criados em atividades educacionais. Essas atividades devem ser realizadas de maneira que os alunos conheçam o desempenho desejado (nível de referência), saibam quão distante o desempenho atual deles está em comparação com o nível de referência e possam agir de maneira a diminuir essa lacuna (SADLER, 1989). No contexto do ensino de computação, a avaliação baseada no desempenho pode ser realizada avaliando um artefato de software, por exemplo, um aplicativo móvel criado em uma atividade de programação de aplicativos móveis.

Avaliações baseadas no desempenho podem ser suportadas pelo uso de rubricas. Rubricas são formadas por um conjunto de critérios (ALLEN; TANNER, 2006) que podem estar organizados em várias categorias, permitindo avaliar o processo, o desempenho e o progresso dos alunos (WHITTAKER; SALEND; DUHANEY, 2001). Rubricas geralmente são apresentadas na forma de uma tabela com uma linha para cada critério e uma coluna para cada nível de desempenho (Figura 18). Os níveis de desempenho compõem uma escala que varia de baixo desempenho ao padrão de desempenho desejado. O cruzamento de cada linha (critério) com cada coluna (nível de desempenho) contém uma descrição caracterizando o nível de desempenho que deve ser útil para julgar o progresso do aluno em relação aos objetivos da atividade (ALLEN; TANNER, 2006).

Figura 18 – Elementos de uma rubrica.




| Item | 0 pontos | 1 ponto | 2 pontos | 3 pontos |
|---|---|---|---|---|
| **Eventos**: verificar se são usados eventos. | Nenhum manipulador de evento é usado (ex. On click). | 1 tipo de manipuladores de eventos é usado. | 2 tipos de manipulador es de eventos são usados. | Mais de 2 tipos de manipuladores de eventos são usados. |
| **Laços**: verificar se são usados laços. | Não usa laços | Usa "While" (laço simples) | Usa "For each" (variável simples) | Usa "For each" (item de lista) |
| **Condicionais**: verificar se são usados condicionais. | Não usa condicionais. | Usa apenas "if's" simples. | Usa apenas "if then else". | Usa um ou mais "if - else if". |

Fonte: Alves (2019)

Rubricas têm o benefício de tornar explícito o padrão utilizado para atribuir notas, além de indicar aos alunos o que é necessário para alcançar um alto nível de desempenho (ALLEN; TANNER, 2006). Elas auxiliam os alunos em entender as qualidades associadas a uma tarefa e avaliar o seu próprio trabalho. Além disso, rubricas podem tornar a atribuição de notas mais objetiva e consistente (WHITTAKER; SALEND; DUHANEY, 2001). Uma nota pode ser determinada com base na pontuação atingida nos critérios da rubrica, por exemplo, somando as pontuações dos critérios, possivelmente atribuindo pesos aos critérios.

**Automatização da avaliação.** A avaliação da qualidade de aplicativos no contexto de ensino de computação pode ser automatizada utilizando diversas abordagens. Nesse contexto, a análise dinâmica tem o objetivo de encontrar erros em um programa executando-o com certos conjuntos de dados por meio de testes automatizados de software. A análise estática, por outro lado, tenta identificar problemas analisando o código do programa sem executá-lo (TRUONG; ROE; BANCROFT, 2004).

No contexto de atividades abertas de desenvolvimento de aplicativos, não há uma solução correta para se comparar por meio de análise dinâmica com as saídas do aplicativo desenvolvido, inviabilizando a adoção desse tipo de abordagem de automação. Por outro lado, a análise estática de código é usada por ferramentas automatizadas para avaliar conceitos de programação em atividades dessa natureza (ALVES; GRESSE VON WANGENHEIM; HAUCK, 2019). Nesse caso, a avaliação é automatizada utilizando diversas linguagens, como Python e Java, e a ferramenta de avaliação é disponibilizada como um sistema *web* ou uma aplicação *desktop*, ou pode ser acessada por meio de scripts.

Especificamente para aplicativos criados com o App Inventor, é possível automatizar também a avaliação do design visual, além de conceitos de programação, por meio de análise estática do código. Isso é possível uma vez que o App Inventor permite realizar o design da interface dos aplicativos desenvolvidos e o registra como parte do código fonte.

# 3 ESTADO DA ARTE

Este capítulo apresenta um mapeamento sistemático da literatura com o objetivo de levantar o estado da arte sobre abordagens que avaliam o desempenho do aluno em relação ao design de interface em atividades de programação de aplicativos móveis. O mapeamento segue a metodologia proposta por Petersen, Vakkalanka e Kuzniarz (2015). O mapeamento responde à pergunta de pesquisa: Quais meios existem para a avaliação do design visual da interface de aplicativos móveis para *smartphones touchscreen* Android (desenvolvidos com App Inventor)? O foco principal do mapeamento está na avaliação de aplicativos desenvolvidos com o App Inventor, o ambiente de programação mais popular para o desenvolvimento de aplicativos móveis na educação básica. No entanto, são consideradas abordagens de avaliação de aplicativos Android em geral (referidas como "abordagens gerais" no restante deste capítulo), além das abordagens com foco educacional ("abordagens educacionais"), para se obter uma visão mais ampla da área.

## 3.1 DEFINIÇÃO DO PROTOCOLO DO MAPEAMENTO

A pergunta de pesquisa foi decomposta nas seguintes perguntas de análise:

PA1. Quais modelos/ferramentas de avaliação do design visual existem?

PA2. Quais as características do modelo de avaliação do design visual?

PA3. Quais as características instrucionais do modelo/ferramenta?

PA4. Como o modelo/ferramenta foi desenvolvida?

PA5. Como a qualidade do modelo/ferramenta foi avaliada?

**Bases de dados.** As buscas foram realizadas nas principais bases de dados e bibliotecas digitais da área da computação: ACM Digital Library, IEEE Xplore Digital Library, ScienceDirect, SpringerLink, Wiley e Scopus. Além dessas bases de dados, com o intuito de abranger uma maior gama de publicações, foram realizadas buscas no Google Scholar, que indexa um grande conjunto de dados de diversas fontes de produção científica (HADDAWAY et al., 2015).

Buscas informais foram utilizadas para calibrar a *string* de busca, a qual foi definida com base nos termos relevantes da pergunta de pesquisa do mapeamento. Durante essas buscas fora do protocolo do mapeamento, notou-se a existência de poucos artigos relevantes para o foco específico da pesquisa. Portanto, decidiu-se ampliar o escopo das buscas, resultando nos termos de busca, sinônimos e termos similares apresentados na Tabela 5.

Tabela 5 – Termos de busca relevantes e sinônimos ou termos similares.

| **Termo** | **Sinônimos ou termos similares** |
|---|---|
| App Inventor | Android |
| Grading | Assessment |
| Usability | User experience, ux, user interface, design |

Fonte: Elaborada pelo autor

Assim, definiu-se a seguinte *string* de busca, a qual foi adaptada para cada base de dados conforme a sua sintaxe:

```
("app inventor" OR android) AND (grading OR assessment) AND (app OR "mobile application") AND ("user experience" OR ux OR "user interface" OR design OR usability)
```

**Critérios de inclusão e exclusão.** Os artigos analisados foram mantidos ou descartados conforme os seguintes critérios de inclusão e exclusão:

- Incluir artigos científicos e artefatos que apresentam *checklists* ou rubricas que avaliam a qualidade do design de interface de aplicativos móveis;
- Incluir apenas artigos em inglês ou português, e excluir artigos em outros idiomas;
- Incluir apenas artigos acessíveis via Portal CAPES;
- Incluir artigos que apresentam critérios de avaliação do design de interface, não limitando a abordagens para uso educacional para se ter uma visão mais ampla do estado da arte.

**Critérios de qualidade.** Foram incluídos apenas artigos contendo informações substanciais indicando como o design visual é avaliado. Ou seja, os critérios de avaliação devem estar explícitos.

## 3.2 EXECUÇÃO DA BUSCA

A busca inicial, executada nos meses de maio e junho de 2018, resultou em mais de 20,000 artigos. Portanto, nas bases de dados SpringerLink, Wiley e Scopus, os resultados foram filtrados, e apenas artigos das áreas de Ciência da Computação e Educação foram analisados. Apenas as 200 publicações mais relevantes do Google Scholar foram analisadas. Todos os artigos encontrados nas bases ScienceDirect e ACM Digital Library foram analisados. A IEEE Xplore Digital Library não retornou nenhum resultado.

Os artigos foram analisados em três etapas, nas quais artigos foram excluídos ou mantidos conforme os critérios de inclusão e exclusão:

- Artigos fora do escopo foram excluídos com base no título, resumo e palavras-chave;
- O resumo dos artigos mantidos na etapa 1 foi lido novamente, e outros elementos foram rapidamente analisados, como as conclusões;
- Os artigos restantes da etapa 2 foram analisados na íntegra.

O número de artigos mantidos em cada etapa para cada base de dados é apresentado na Tabela 6. Ao final, identificaram-se 7 artigos relevantes. Muitos artigos encontrados pelas buscas foram excluídos porque tinham foco em questões de segurança, em propor um aplicativo, ou em avaliar aplicativos, mas sem tratar o design de interface. Foram descartados também vários artigos que relatavam o desenvolvimento de aplicativos móveis em contexto educacional, mas não apresentavam a avaliação dos aplicativos.

Tabela 6 – Número de artigos mantidos em cada etapa.

| Base | Artigos encontrados | Artigos analisados | 1ª etapa | 2ª etapa | 3ª etapa |
|---|---|---|---|---|---|
| SpringerLink | 950 | 314 | 51 | 10 | 0 |
| ScienceDirect | 647 | 647 | 78 | 19 | 2 |
| Wiley | 565 | 160 | 21 | 1 | 0 |
| IEEE | 0 | 0 | 0 | 0 | 0 |
| ACM | 42 | 42 | 9 | 2 | 1 |
| Scopus | > 1900 | 1239 | 63 | 22 | 3 |
| Google Scholar | > 1700 | 200 | 17 | 9 | 1 |
| **Total** | **> 20000** | **2571** | **239** | **63** | **7** |

Fonte: Desenvolvida pelo autor

Ao final, foram incluídos mais dois artefatos (TECHNOVATION, 2014) (GOOGLE, 2018a), que foram encontrados apenas em buscas informais por não serem artigos científicos, mas satisfazem os critérios de inclusão e exclusão definidos.

## 3.3 EXTRAÇÃO DOS DADOS

Para responder às perguntas de análise, foram extraídos de cada publicação os dados apresentados na Tabela 7.

Tabela 7 – Dados extraídos.

| Pergunta de análise | Dados extraídos |
|---|---|
| PA1. Quais modelos / ferramentas de avaliação do design visual existem? | Nome |
| | Referência |
| PA2. Quais as características do modelo de avaliação do design visual? | Critérios de design de interface avaliados |
| | Em que é baseado (norma, guia de estilo etc.) |
| | Tipo do modelo (*checklist* ou rubrica) |
| | A avaliação é automatizada? |
| | Contexto de aplicação (tipos de aplicativos) |
| PA3. Quais as características instrucionais do modelo/ferramenta? | É usado em contexto educacional? |
| | Descrição geral da unidade instrucional |
| | Ambiente de programação usado na unidade instrucional |
| | Língua(s) em que a unidade instrucional está disponível |
| | Nível escolar para o qual a unidade instrucional foi projetada |
| PA4. Como o modelo/ferramenta foi desenvolvido? | Método de desenvolvimento da unidade instrucional |
| PA5. Como a qualidade do modelo/ferramenta foi avaliada? | Tipo de estudo |
| | Fatores avaliados |
| | Método de coleta de dados |
| | Tamanho da amostra |
| | Método de análise |
| | Descobertas |

Fonte: Elaborada pelo autor

## 3.4 RESULTADOS DO MAPEAMENTO

Esta seção apresenta uma síntese dos resultados, respondendo a cada uma das perguntas de análise.

### 3.4.1 Quais modelos/ferramentas de avaliação do *design* visual existem?

No total, foram encontradas nove publicações que contêm alguma forma de avaliação do design visual de interface de aplicativos móveis para *smartphones touchscreen* Android ou desenvolvidos com o App Inventor. Essas publicações são listadas na Tabela 8.

Tabela 8 – Abordagens encontradas como resultado do mapeamento.

| Autoria | Título |
|---|---|
| *Abordagens gerais* | |
| Hoehle, Aljafari e Venkatesh (2016) | Leveraging Microsoft's mobile usability guidelines: Conceptualizing and developing scales for mobile application usability |
| Ines et al. (2017) | Evaluation of Mobile Interfaces as an Optimization Problem |
| Zapata et al. (2014) | Mobile PHRs Compliance with Android and iOS Usability Guidelines |
| Gresse von Wangenheim et al. (2016) | A usability score for mobile phone applications based on heuristics |
| Google (2018a) | Core app quality |
| *Abordagens educacionais* | |
| Sherman e Martin (2015) | The assessment of mobile computational thinking |
| Gresse von Wangenheim et al. (2018) | CodeMaster – Automatic Assessment and Grading of App Inventor and Snap! Programs |
| Wagner et al. (2013) | Using app inventor in a K–12 summer camp |
| Technovation (2014) | Teacher and Mentor Lesson Guide Lessons 1 – 6 |

Fonte: Elaborada pelo autor

Tabela 9 – Tipo dos critérios de design visual em cada abordagem.

| Autoria | Tipo dos critérios | | |
|---|---|---|---|
| | Estilo | Componentes | Subjetivo |
| *Abordagens gerais* | | | |
| Hoehle, Aljafari e Venkatesh (2016) | x | | |
| Ines et al. (2017) | x | | |
| Zapata et al. (2014) | x | | |
| Gresse von Wangenheim et al. (2016) | x | | |
| Google (2018a) | x | | |
| *Abordagens educacionais* | | | |
| Sherman e Martin (2015) | | x | |
| Gresse von Wangenheim et al. (2018) | | x | |
| Wagner et al. (2013) | | | x |
| Technovation (2014) | | | x |

Fonte: Elaborada pelo autor

### 3.4.2 Quais as características do modelo de avaliação do design visual?

O foco dos critérios de avaliação do design visual varia entre as abordagens encontradas. Portanto, as abordagens foram classificadas em exatamente uma das seguintes categorias (Tabela 9):

**Estilo:** Contém múltiplos critérios, cada um relacionado a uma questão específica de estilo (como tamanho, cor ou posicionamento de controles e informações) ou do uso correto de texto e imagens/ícones;

**Componentes:** Possui critérios relacionados ao uso de componentes – quantos componentes foram usados e como foram usados;

**Subjetivo:** Possui critérios pouco específicos, que não se encaixam nas outras categorias (Estilo e Componentes).

Todas as abordagens gerais (5 de 9) avaliam critérios relacionados a estilo. Critérios desse tipo são similares às diretrizes de guias de estilo de plataforma, e observa-se que algumas dessas abordagens realmente baseiam-se nelas. Por outro lado, nenhuma das abordagens educacionais possui critérios com esse foco, avaliando apenas o uso de componentes visuais (2 de 9) ou a interface como um todo sem critérios detalhados (2 de 9). Observa-se, portanto, que todas as abordagens educacionais contam com uma avaliação muito limitada do *design* de interface, focando principalmente em conceitos de programação.

Quanto ao instrumento de avaliação, aproximadamente metade das abordagens (5 de 9) utilizam *checklists*, enquanto as demais (4 de 9) utilizam rubricas. Poucas são suportadas por avaliação automatizada (2 de 9). Quanto à origem dos critérios, em geral as abordagens baseiam-se em diretrizes de plataforma, trabalhos similares e heurísticas de usabilidade. Dois trabalhos (WAGNER et al., 2013) (TECHNOVATION, 2014) não informam com base em que os critérios foram desenvolvidos, e o trabalho de Sherman e Martin (2015) relata a experiência dos autores analisando trabalhos de alunos como base para derivar os critérios.

### 3.4.3 Quais as características instrucionais do modelo/ferramenta?

Todas as abordagens educacionais envolvem a criação de aplicativos utilizando o App Inventor. Somente a abordagem de Wagner et al. (2013) envolve o uso de outro ambiente de programação, utilizando também a biblioteca de software Java Bridge como forma de auxiliar os alunos a fazerem uma transição do desenvolvimento com o App Inventor para a linguagem Java. A abordagem de Sherman e Martin (2015) é utilizada em nível de graduação, enquanto as demais são direcionadas à educação básica. O único modelo suportado por automação entre as abordagens educacionais é o CodeMaster v1.0 (GRESSE VON WANGENHEIM et al., 2018), apresentando uma ferramenta que avalia um projeto do App Inventor segundo a rubrica definida no modelo e apresenta a nota para cada critério, além de uma nota final. Todas as unidades instrucionais foram disponibilizadas na língua inglesa, e apenas a ferramenta CodeMaster está disponível também em outra língua, Português Brasileiro (Tabela 10).

Tabela 10 – Características instrucionais das abordagens educacionais.

| **Abordagem** | **Descrição** | **Ambiente de programação** | **Língua(s)** | **Nível escolar** |
|---|---|---|---|---|
| Sherman e Martin (2015) | Rubrica de pensamento computacional incluindo um critério sobre a interface e uma pergunta subjetiva sobre a estética | App Inventor | Inglês | Graduação |
| Gresse von Wangenheim et al. (2018) | Rubrica de pensamento computacional incluindo dois critérios que avaliam quantos componentes foram utilizados na interface | App Inventor | Inglês; Português | Educação Básica |
| Wagner et al. (2013) | Acampamento de verão de 3 semanas. Na terceira, introduz-se o App Inventor, e em seguida Java Bridge, uma biblioteca Java com os componentes do App Inventor | App Inventor; Java com Java Bridge e IDE eclipse | Inglês | Ensino Médio |
| Technovation (2014) | Programa para meninas envolvendo o desenvolvimento de um aplicativo móvel, com o objetivo de inspirá-las a entrarem no ramo da tecnologia | App Inventor | Inglês | Ensino Médio |

Fonte: Elaborada pelo autor

Tabela 11 – Método de desenvolvimento das abordagens.

| **Abordagem** | **Método de desenvolvimento** |
|---|---|
| Hoehle, Aljafari e Venkatesh (2016) | O autor revisou as diretrizes da Microsoft e codificou o conteúdo usando a análise linha-a-linha de Strauss e Corbin (1990), organizou os códigos usando codificação axial e uma matriz conforme Miles e Huberman (1984). Outro autor revisou as diretrizes, e discordâncias entre os autores foram discutidas para chegar a um consenso. |
| Ines et al. (2017) | A lista de métricas usadas e defeitos detectados foram derivados de outros trabalhos encontrados na literatura. |
| Zapata et al. (2014) | O questionário foi desenvolvido com base em: (i) padrões e recomendações de usabilidade, (ii) literatura relacionada e (iii) as diretrizes oficiais de design para Android e iOS. |
| Gresse von Wangenheim et al. (2016) | A abordagem GQM (Goal, Question, Metric) foi utilizada para definir métricas sistematicamente. O conjunto inicial de itens foi definido de maneira abrangente, conforme Clark e Watson (1995). |
| Google (2018a) | *Não informado (mas faz referência às diretrizes da plataforma Android)* |
| Sherman e Martin (2015) | Com base nas funcionalidades disponíveis no App Inventor e análise de trabalhos de alunos, os critérios de avaliação foram ajustados para cobrirem os conceitos usados nos trabalhos dos alunos. |
| Gresse von Wangenheim et al. (2018) | A rubrica foi desenvolvida seguindo o procedimento proposto por Andrade (1997). Os critérios de avaliação foram definidos com base em (i) estruturas e rubricas existentes e estratégias instrucionais para a educação de computação na educação básica e (ii) os recursos específicos dos ambientes de programação baseados em blocos (App Inventor e Snap!). A ferramenta CodeMaster foi desenvolvida usando uma abordagem iterativa incremental. |
| Wagner et al. (2013) | *Não informado* |
| Technovation (2014) | *Não informado* |

Fonte: Elaborada pelo autor

### 3.4.4 Como o modelo/ferramenta foi desenvolvido(a)?

O uso de um método sistemático para desenvolver um modelo de avaliação pode auxiliar na definição de critérios apropriados, que medem a aprendizagem de forma adequada. A maior parte das abordagens educacionais utiliza rubricas, que devem refletir um padrão de desempenho satisfatório (ALLEN; TANNER, 2006), porém definir padrões de desempenho não é uma tarefa trivial (SADLER, 1989). Além disso, *feedback* deve ser planejado e fornecido de maneira apropriada para auxiliar os estudantes a atingirem esse padrão de desempenho. O método de desenvolvimento extraído das publicações é apresentado na Tabela 11.

Observa-se que, em geral, as abordagens foram desenvolvidas seguindo metodologias encontradas na literatura, cada uma seguindo uma metodologia distinta. No entanto, algumas das publicações não relatam o processo de desenvolvimento. Apenas um terço das abordagens baseia-se em diretrizes de guias de estilo de plataforma.

Tabela 12 – Avaliação das abordagens.

| Abordagem | Tipo de estudo | Fatores avaliados | Método de coleta de dados | Tamanho da amostra | Método de análise | Descobertas |
|---|---|---|---|---|---|---|
| Hoehle, Aljafari e Venkatesh (2016) | *Survey* | Validade de conteúdo;<br>Estrutura fatorial;<br>Propriedades da escala; | 4 questionários | 30;<br>318;<br>404;<br>501 | PSA, CSV (ANDERSON; GERBING, 1991)<br>Análise fatorial exploratória;<br>Análise fatorial confirmatória | Princípios da Gestalt são críticos em aplicativos móveis |
| Ines et al. (2017) | Comparação entre avaliação automática e manual | Corretude | Questionário | 20 | Precisão e *recall* | Em média, houve mais de 70% de precisão e *recall* |
| Zapata et al. (2014) | *Survey* | Aceitação;<br>Concordância | Questionário | 22 | Cálculo da<br>(i) porcentagem de aceitação;<br>(ii) média/desvio padrão | A média de aceitação foi de 71,2% |
| Gresse von Wangenheim et al. (2016) | Estudo empírico | Validade | Avaliação heurística | 247 | TRI | Algumas heurísticas tradicionais não são relevantes para aplicativos móveis, enquanto as novas heurísticas são |
| Google (2018a) | *Não informado* | | | | | |
| Sherman e Martin (2015) | Comparação com "resultados esperados" | Eficácia | Análise manual do código e do aplicativo em execução | 18 | ANOVA | A rubrica é capaz de discriminar estudantes de computação de estudantes de outras áreas |
| Gresse von Wangenheim et al. (2018) | Testes de usuários e de corretude | Qualidade do ponto de vista de alunos e professores da educação básica;<br>Corretude | Observação<br>Questionário | 16 | Estatística descritiva | Representação lúdica da nota agrada os alunos;<br>Obter *feedback* motiva os alunos a continuarem programando;<br>A terminologia precisa ser revisada |
| Wagner et al. (2013) | *Não informado* | | | | | |
| Technovation (2014) | *Não informado* | | | | | |

Fonte: Elaborada pelo autor

### 3.4.5 Como a qualidade do modelo/ferramenta foi avaliada?

Modelos de avaliação devem ser avaliados e consequentemente refinados para que pesquisadores, professores e outros interessados possam usá-los tendo evidências de que o instrumento é eficaz (WHITTAKER et al., 2001). A avaliação de cada abordagem é apresentada na Tabela 12. Em geral, as abordagens foram avaliadas de forma bem definida, utilizando técnicas estatísticas bem estabelecidas para suportar as conclusões sobre a qualidade. No entanto, algumas publicações não relatam nenhuma avaliação, incluindo metade das abordagens com fins educacionais (2 de 4).

## 3.5 DISCUSSÃO

Os resultados revelam que existem poucas abordagens de avaliação de aplicativos móveis Android incluindo a avaliação do design visual. Essa falta se agrava uma vez que a avaliação é pouco detalhada nas abordagens existentes, especialmente no contexto educacional, visto que não são abordados de forma específica e detalhada os principais elementos do design visual (*layout*, cores etc.). As abordagens também não fornecem *feedback* quanto ao design visual de forma detalhada, o qual é essencial, pois permite melhorar a aprendizagem quando fornecido de forma construtiva.

As abordagens gerais avaliam o design visual de forma mais detalhada que as educacionais. No entanto, como não possuem foco educacional, não fornecem o *feedback* necessário para o ciclo de aprendizagem, não apresentando os resultados da avaliação especificamente em relação aos elementos e princípios de design visual. Além disso, constata-se a falta de suporte em geral à avaliação automatizada.

Embora este mapeamento não tenha sido restringido a métodos de avaliação de aplicativos criados com o App Inventor, todas as abordagens educacionais encontradas que avaliam o design visual envolvem o uso significativo ou exclusivo do App Inventor. Isso é possível pois o App Inventor dá suporte ao design visual dos aplicativos desenvolvidos.

Conclui-se que atualmente falta suporte ao ensino de design visual de aplicativos móveis Android (criados com o App Inventor). Para fornecer esse suporte, é necessário um modelo de avaliação que aborde os elementos centrais do design visual (*layout*, cores etc.) de forma detalhada, considerando os princípios de design visual e diretrizes de guias de estilo. Além disso, é necessário que se forneça *feedback* construtivo, o que é essencial no ciclo de aprendizagem. Por fim, para viabilizar o ensino de design visual no contexto da educação básica, é necessário que o modelo de avaliação do design visual seja suportado por técnicas de avaliação automatizada.

## 3.6 AMEAÇAS À VALIDADE

Mapeamentos sistemáticos podem ter resultados tendenciosos pois resultados positivos têm maior chance de serem publicados do que resultados negativos. No entanto, no caso deste mapeamento, os resultados das publicações não têm influência significativa, visto que o objetivo principal é caracterizar as abordagens existentes.

Outro risco é o de omitir estudos relevantes. Para mitigá-lo, a *string* de busca foi construída buscando utilizar todos os termos relevantes e incluindo sinônimos. Além disso, a busca foi realizada em diversas bases de dados científicas, reduzindo as chances de omitir estudos relevantes.

Para mitigar riscos na seleção dos estudos e extração de dados, foi utilizada uma definição detalhada dos critérios de inclusão e exclusão. Um protocolo de seleção foi definido e executado pelo autor deste trabalho em conjunto com a aluna de graduação em Ciência da Computação Karla Aparecida Justen, e os resultados foram posteriormente revisados pela orientadora do autor deste trabalho.

# 4 DESENVOLVIMENTO DO CODEMASTER UI DESIGN – APP INVENTOR

Este capítulo apresenta o desenvolvimento da abordagem, incluindo a análise do contexto, a definição da rubrica CodeMaster UI design – App Inventor e a automatização da rubrica, evoluindo a ferramenta de software CodeMaster v1.0.

## 4.1 ANÁLISE DO CONTEXTO

Seguindo o modelo ADDIE de design instrucional (BRANCH, 2009), a etapa inicial é a análise do contexto em que a abordagem será aplicada. Nesse contexto, as competências relacionadas à computação estão entre as que devem ser desenvolvidas pelos alunos desde a Educação Básica, de forma que o ensino de computação está mundialmente sendo inserido em currículos da Educação Básica (CSTA, 2016) (BNCC, 2018). Assim, são necessárias formas de avaliação adequadas para esse contexto. Portanto, a abordagem proposta é direcionada ao contexto da Educação Básica, tendo como público alvo alunos do 3º ano do Ensino Fundamental ao 3º ano do Ensino Médio, na faixa etária de 8 a 17 anos, conforme os currículos do CSTA (2016) e do MEC (2017).

Segundo o *framework* para ensino de computação na Educação Básica, um dos conceitos transversais que fazem parte do ensino de computação é a Interação Humano-Computador, que abrange o design de IU e, mais especificamente, o design visual. O ensino do design de interface está alinhado também às práticas do *framework* (ver Figura 14), que definem as competências que os alunos devem ter desenvolvido ao final do Ensino Médio, incluindo:

- **Criar artefatos computacionais:** Os alunos devem criar artefatos (p. ex., aplicativos móveis) relevantes para si mesmos ou benéficos para a comunidade;
- **Testar e refinar artefatos computacionais:** Os alunos devem considerar as necessidades de usuários finais para melhorar os artefatos, incluindo a usabilidade e acessibilidade.

Focando principalmente em conceitos de algoritmos e programação, existem diversas unidades instrucionais voltadas ao desenvolvimento de software como jogos, robótica ou aplicativos móveis (GARNELI; GIANNAKOS; CHORIANOPOULOS, 2015). Porém, visto que o design de IU é essencial no desenvolvimento de um aplicativo móvel, é importante também o desenvolvimento de competências relacionadas ao design de IU como parte do aprendizado de computação (SOLECKI et al., 2020). Assim, já existem várias unidades instrucionais que abordam também o ensino de competências de design de IU (FERREIRA et

al., 2020; FERREIRA et al., 2019b). Nessas unidades, tipicamente se ensina computação e competências de design de IU por meio de atividades abertas em que os alunos desenvolvem aplicativos móveis para lidar com problemas na sua comunidade usando, por exemplo, o App Inventor (MIT, 2019). Assim, há um foco nos efeitos sociais dos aplicativos desenvolvidos, de forma alinhada à perspectiva da ação computacional (PATTON; TISSENBAUM; HARUNANI, 2019).

Como parte do processo de aprendizagem, é necessário avaliar o aprendizado de competências de design de IU. Uma forma possível é a avaliação de desempenho por meio da avaliação de aplicativos móveis criados pelos alunos em atividades abertas.

Com base nas competências relacionadas ao design de IU que devem ser desenvolvidas na Educação Básica (CSTA, 2016), e tendo em vista a falta de uma definição estabelecida na literatura de objetivos de aprendizagem relacionados ao design visual, definiu-se o seguinte objetivo de aprendizagem para unidades instrucionais para o ensino de design visual:

**Objetivo geral:** Criar um design visual de interface de usuário com boa estética visual.

Esse objetivo geral foi decomposto nos seguintes objetivos específicos:

OA1. Adequar o design ao contexto de uso

OA2. Projetar interfaces minimalistas e estéticas

OA3. Projetar interfaces consistentes, seguindo padrões da plataforma

OA4. Apresentar texto e imagens com legibilidade

O objetivo OA1 reflete a etapa de análise de requisitos no processo de Engenharia de Usabilidade (ROGERS; SHARP; PREECE, 2013), que envolve a caracterização dos usuários, bem como do dispositivo, o que é essencial para um design visual apropriado. Os objetivos OA2 e OA3 são derivados das heurísticas de Nielsen (1994), estando alinhados aos princípios do design visual. O objetivo OA4 baseia-se em orientações sobre legibilidade presentes em diretrizes de acessibilidade, como as WCAG 2.1 (W3C, 2018). Assim, esses objetivos específicos estão alinhados aos objetivos de aprendizagem do guia de currículo para a Educação Básica do CSTA (2019), os quais indicam que os alunos devem desenvolver competências para considerar a usabilidade e acessibilidade ao desenvolver e avaliar aplicativos.

Nas escolas de Educação Básica, a computação frequentemente é ensinada apenas de forma interdisciplinar ou extracurricular, visto que a computação não é abordada explicitamente pelo currículo do MEC (2017). Outro fator que dificulta o ensino de computação de forma ampla na Educação Básica é a falta de professores com formação de computação (INEP, 2017), de forma que professores formados em outras áreas ministram o conteúdo de computação. Além

disso, as atividades de avaliar e fornecer *feedback* a todos os alunos requerem um esforço grande devido ao tamanho das turmas, o que se agrava devido à carga horária elevada em sala de aula de professores da Educação Básica.

## 4.2 DEFINIÇÃO DA AVALIAÇÃO

A abordagem tem o objetivo de suportar uma avaliação baseada em desempenho do design visual de aplicativos móveis. A avaliação é realizada a partir da análise de um aplicativo criado pelo aluno como parte de uma unidade instrucional para o ensino de design visual como parte do ensino de computação. A partir dos objetivos de aprendizagem referentes a competências de design de IU no nível educacional da Educação Básica detalhados na Seção 4.1, são derivados os critérios de desempenho especificados na Tabela 13. Esses critérios baseiam-se também na teoria de design visual e em diretrizes de guias de estilo e de acessibilidade (SCHLATTER; LEVINSON, 2013; GOOGLE, 2019; W3C, 2018).

Tabela 13 – Descrição dos critérios e objetivos de aprendizagem.

| **Critério** | **Descrição** | **Objetivos de aprendizagem** |
|---|---|---|
| L1 | Componentes que são alvo de toque devem ter altura e largura de no mínimo 48dp. | OA1 |
| L2 | O App Inventor apresenta as opções de botão: padrão (com degradê); retangular; oval; e com bordas arredondadas. Para satisfazer o princípio de consistência, todos os botões devem possuir o mesmo formato. | OA3 |
| L3 | Botões que estão dentro do mesmo elemento de organização devem possuir o mesmo tamanho, de forma que fiquem alinhados. | OA2 |
| L4 | As telas do aplicativo não devem ser muito vazias nem muito cheias de elementos visuais. Assim, com base na observação de aplicativos criados com o App Inventor, considera-se ideal que as telas possuam de 2 a 9 elementos visuais. | OA1 OA2 |
| T1 | A fonte usada no aplicativo deve ser sem serifa. O App Inventor não disponibiliza a família de fonte indicada pelo Material Design (Roboto). Assim, a alternativa mais próxima é "sem serifa". A opção "padrão" também pode ser usada, pois geralmente corresponde a "sem serifa". | OA4 |
| T2 | Botões devem usar o tamanho de fonte indicado pelo Material Design (14). | OA3 |
| T3 | Todos os componentes com texto (exceto botões, abordados pelo critério T2) devem ter um dos tamanhos de fonte indicados pelo Material Design (10, 12, 14, 16, 20, 24, 34, 48, 60, 96). | OA3 |
| T4 | O estilo itálico em geral deve ser usado para destaque de apenas algumas palavras. Assim, os componentes não devem possuir o texto completamente em itálico. | OA4 |
| E1 | Botões devem possuir o texto completamente em letras maiúsculas, conforme o estilo indicado pelo Material Design para botões. | OA3 |
| E2 | Sentenças não devem começar com letra minúscula. | OA4 |
| E3 | Ao criar componentes no App Inventor, eles contêm um texto padrão (p. ex., "Texto para Botão1"). Esse texto padrão não transmite nenhuma mensagem significativa e, portanto, não deve ser mantido. | OA3 |
| E4 | Dois pontos (:) devem ser evitados ao final de legendas. | OA2 |
| E5 | Ponto final (.) deve ser evitado ao final de frases isoladas.<br>**Nota:** Frases podem terminar com reticências (…), conforme exemplos do Material Design. | OA2 |
| E6 | O texto de botões deve ser conciso. Para pontuar no critério, espera-se que o comprimento em caracteres seja de até 7 caracteres, ou até 14 para receber pontuação parcial. | OA2 |

| | | |
|---|---|---|
| C1 | Visto que cores podem tornar a interface mais atrativa, espera-se que haja pelo menos uma cor (desconsiderando tons de cinza). No entanto, a quantidade de cores deve ser limitada, idealmente a no máximo uma cor primária e uma secundária.<br>**Nota:** Cores de imagens são consideradas neste critério. | OA2 OA3 |
| C2 | Para garantir a acessibilidade, o texto deve ter contraste suficiente, conforme os limites definidos pelas WCAG 2.1. | OA1 OA4 |
| C3 | Um sistema de cores agradável pode ser definido mais facilmente utilizando as cores da paleta do Material Design. | OA2 OA3 |
| C4 | As cores usadas no aplicativo devem satisfazer uma das relações de harmonia (variante, complementar, análoga ou triádica).<br>**Nota:** São aceitas todas as variantes definidas pela paleta do Material Design, embora algumas não seriam consideradas variantes pela posição no círculo cromático. Este critério *não* avalia cores de imagens devido à dificuldade de harmonizar cores de imagens. | OA2 |
| I1 | Os ícones usados no aplicativo devem ser do catálogo de ícones do Material Design.<br>**Nota:** São consideradas como ícones quaisquer imagens usadas em botões. Uma definição mais precisa, diferenciando imagens que são ícones de outros tipos de imagens, não foi possível devido a dificuldades na implementação. | OA3 |
| I2 | Imagens não devem ser pixelizadas.<br>**Nota:** Uma imagem é considerada pixelizada se aparece na tela com largura ou altura 50% maior do que seu tamanho original. Esse valor foi escolhido com base na observação dos efeitos da pixelização. | OA2 OA4 |
| I3 | Imagens não devem ser distorcidas.<br>**Nota:** Uma imagem é considerada distorcida se aparece na tela com largura ou altura pelo menos 10% maior do que deveria para manter a proporção original. Esse valor foi escolhido com base na observação dos efeitos da distorção. | OA2 OA4 |

Fonte: Elaborada pelo autor

A abordagem é instanciada por meio de uma rubrica para a avaliação de projetos criados com o App Inventor, o ambiente de programação baseado em blocos mais popular para o desenvolvimento de aplicativos móveis no contexto da Educação Básica atualmente. A rubrica é apresentada na Tabela 14.

Tabela 14 – Rubrica CodeMaster UI Design – App Inventor.

| **Critério** | **0pt** | **1pts** | **2pts** |
|---|---|---|---|
| *Layout* | | | |
| L1. Todos os componentes alvos de toque têm largura e altura maior ou igual a 48 pixels? | Não | | Sim |
| L2. Todos os botões têm o mesmo formato? | Não | | Sim |
| L3. Botões agrupados na interface sempre têm o mesmo tamanho? | Não | | Sim |
| L4. Qual é o número mínimo e o número máximo de elementos nas telas? | $min < 2$ ou $max \geq 20$ | $min \geq 2$ e $max \in [10, 19]$ | $min \geq 2$ e $max \leq 9$ |
| Tipografia | | | |
| T1. Todos os componentes usam família da fonte sem serifa? | Não | | Sim |
| T2. Todos os botões com texto têm tamanho da fonte igual a 14? | Não | | Sim |
| T3. Todos os componentes (exceto botões) têm um dos tamanhos de fonte recomendados pelo Material Design? | Não | | Sim |
| T4. Há texto em itálico? | Sim | | Não |
| Escrita | | | |
| E1. Todos os botões têm texto todo em letras maiúsculas? | Não | | Sim |
| E2. Todas as sentenças começam com letra maiúscula ou dígito? | Não | | Sim |

| | | | |
|---|---|---|---|
| E3. Todos os componentes têm texto diferente do padrão (p. ex., "Texto para Botão1")? | Não | | Sim |
| E4. Existem legendas que terminam com ":" (dois pontos)? | Sim | | Não |
| E5. Existem sentenças que terminam com "." (ponto)? | Sim | | Não |
| E6. O texto de botão mais longo tem quantos caracteres? | 15 ou mais | De 8 a 14 | 7 ou menos |
| Cores | | | |
| C1. Quantas cores são usadas no aplicativo (além de preto, branco e cinza, incluindo cores de imagens)? | 4 ou mais, ou nenhuma | 3 | 1 ou 2 |
| C2. Qual é o nível WCAG do aplicativo em relação ao contraste do texto? | Insuficiente | Nível AA | Nível AAA |
| C3. Usam-se apenas cores da paleta do Material Design? | Não | | Sim |
| C4. As tonalidades de cores usadas são harmônicas entre si (variantes, complementares, análogas ou triádicas)? | Não | | Sim |
| Imagens | | | |
| I1. Todos os ícones usados são do catálogo de ícones do Material Design? | Não | | Sim |
| I2. Existem imagens pixelizadas? | Sim | | Não |
| I3. Existem imagens distorcidas? | Sim | | Não |

Fonte: Elaborada pelo autor

Considerando o contexto de unidades instrucionais de computação voltadas ao desenvolvimento de aplicativos móveis com o App Inventor, os critérios de avaliação são definidos de acordo com os componentes de IU disponíveis no App Inventor e as possibilidades de configuração da aparência desses componentes. O Material Design recomenda, por exemplo, o uso de *grids* (conjuntos de linhas que permitem alinhar os componentes de forma consistente), porém o App Inventor não dispõe desse mecanismo de alinhamento.

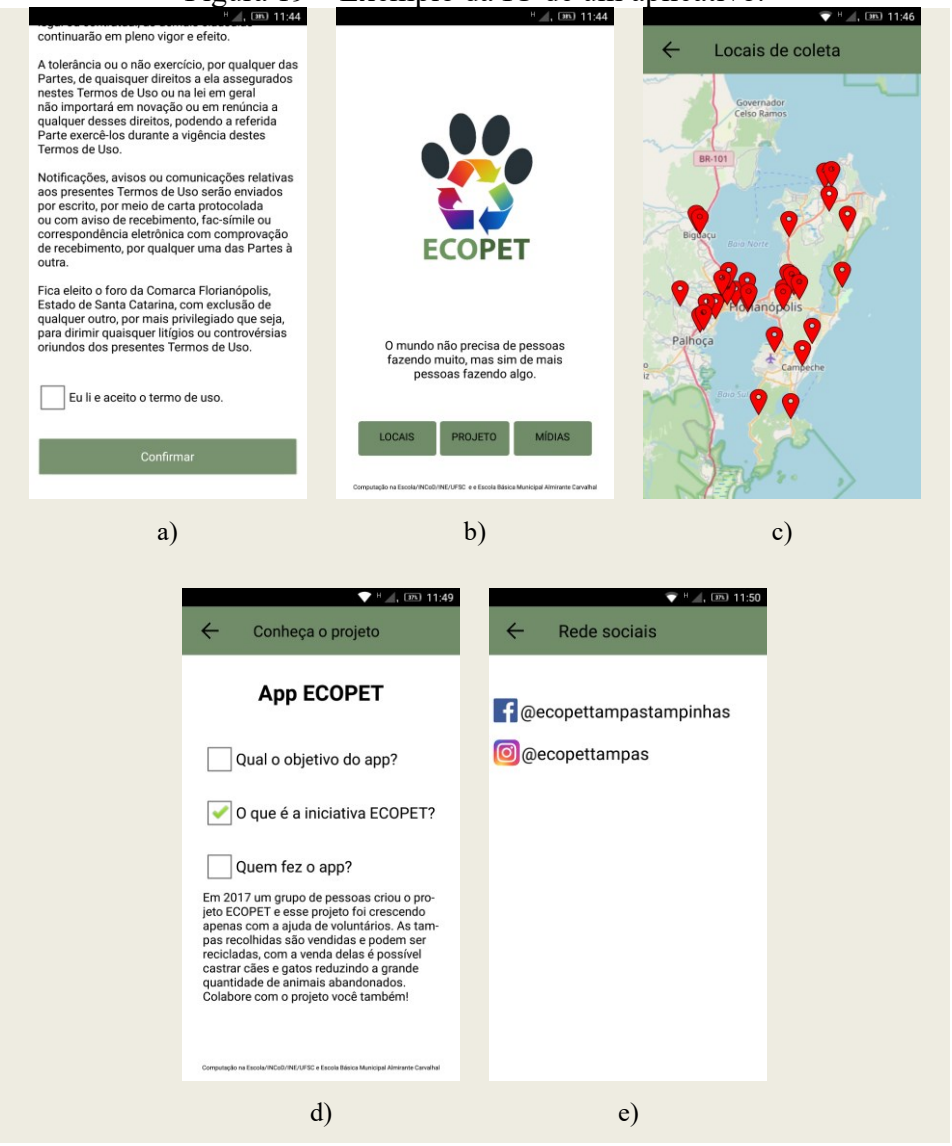
Considerando também o contexto escolar, para minimizar o esforço do professor e manter a qualidade da avaliação, o processo de avaliação foi automatizado. Assim, apenas critérios que podem ser avaliados automaticamente foram considerados. Por exemplo, diretrizes que envolvem semântica são difíceis de automatizar, como a diretriz de que o texto de um botão deve descrever claramente a ação que ele dispara.

Como resultado da avaliação, uma pontuação entre 0 e 2 pontos é atribuída a cada critério. Ao final, uma nota é atribuída ao design visual somando as pontuações dos critérios da rubrica e convertendo em uma nota na escala de 0,0 a 10,0, a escala tipicamente utilizada em escolas brasileiras de Educação Básica:

$$Nota = \frac{pontuação\ atingida}{pontuação\ máxima\ dos\ critérios\ avaliados} \times 10$$

A pontuação atingida é o somatório das pontuações de todos os critérios. Caso o aplicativo sendo avaliado não apresente os componentes que um critério avalia (p. ex., um aplicativo sem imagens), esse critério recebe NA (Não se Aplica) em vez de uma pontuação e não é considerado ao calcular a pontuação atingida e a pontuação máxima.

Figura 19 – Exemplo da IU de um aplicativo.




Fonte: Elaborada pelo autor

Para ilustrar a avaliação utilizando a rubrica, um exemplo de interface de um aplicativo é apresentado na Figura 19, e a sua avaliação, na Tabela 15. Observa-se que o aplicativo obteve pontos em apenas um dos critérios da categoria Cores. A logo (Figura 19 b) contém todo o espectro de cores, resultando em uma quantidade excessiva de cores (critério C1). Além disso, a cor verde do aplicativo não é parte da paleta do Material Design (critério C3), e o botão "Confirmar" (Figura 19 a) possui uma taxa de contraste abaixo do mínimo recomendado (critério C2). Porém, na verificação da harmonia das cores, são consideradas apenas as cores definidas manualmente, de forma que o aplicativo possui apenas uma cor e não apresenta cores fora das relações de harmonia (critério C4).

Tabela 15 – Exemplo de avaliação da rubrica CodeMaster UI Design – App Inventor.

| Item | Pontuação recebida | Pontuação máxima |
|---|---|---|
| L1 tamanhoDeAlvosDeToque | 0 | 2 |
| L2 formatoDosBotoes | 0 | 2 |
| L3 tamanhoConsistenteDeBotoes | 0 | 2 |
| L4 densidadeDaIU | 1 | 2 |
| T1 familiaFonte | 2 | 2 |
| T2 tamanhoFonteBotoes | 0 | 2 |
| T3 tamanhoDaFonte | 0 | 2 |
| T4 evitarItalico | 2 | 2 |
| E1 caixaAltaBotao | 0 | 2 |
| E2 capitalizacaoSentenca | 0 | 2 |
| E3 textoPadrao | 2 | 2 |
| E4 doisPontosLegenda | 2 | 2 |
| E5 pontoFinalSentenca | 0 | 2 |
| E6 tamanhoMaximoTextoBotao | 1 | 2 |
| C1 sistemaDeCores | 0 | 2 |
| C2 contrasteEntreTextoFundo | 0 | 2 |
| C3 coresMaterialDesign | 0 | 2 |
| C4 coresHarmonicas | 2 | 2 |
| I1 iconesMD | 0 | 2 |
| I2 pixelizacao | 2 | 2 |
| I3 distorcao | 2 | 2 |
| **Total** | **16** | **42** |
| **Nota** | **3,81** | **10,0** |

Fonte: Elaborada pelo autor

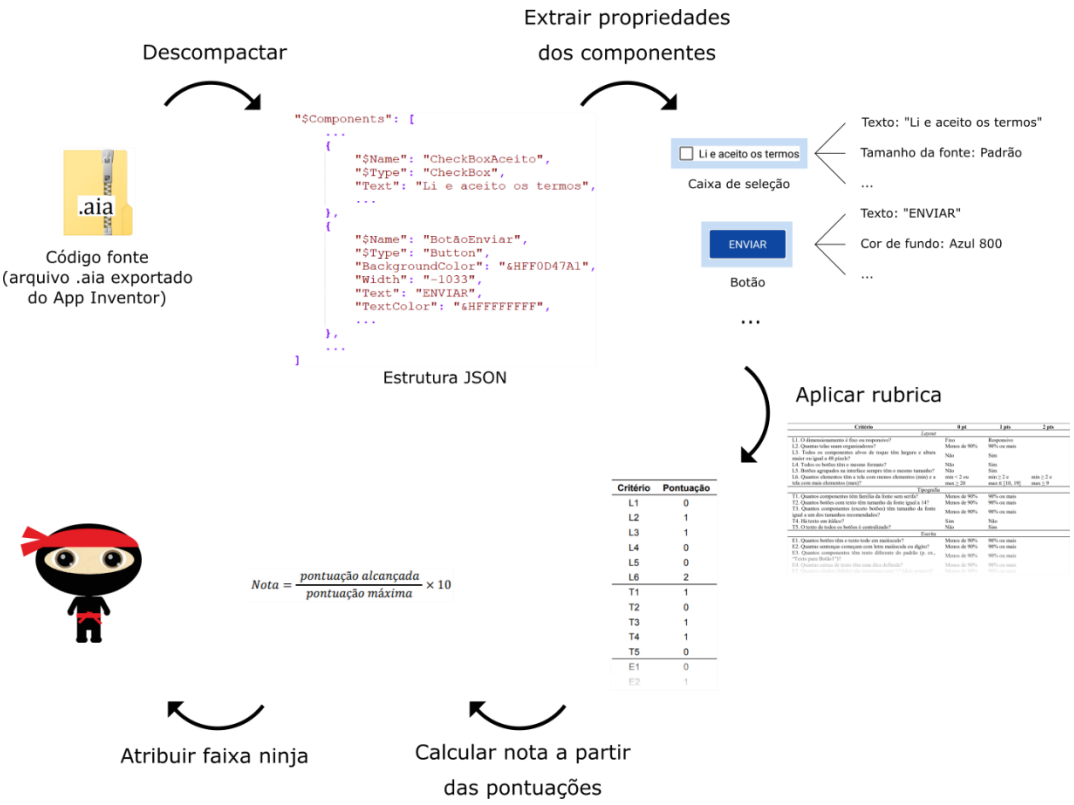
Como *feedback* instrucional para o aluno, a pontuação em cada critério é apresentada (ou NA se o critério não se aplica), bem como uma nota parcial para cada categoria e a nota geral do design visual. Além disso, por meio de uma média ponderada entre a nota do design visual e a nota da rubrica já existente de pensamento computacional, a nota final do aplicativo é determinada, e uma representação lúdica dessa nota final é apresentada na forma de um nível ninja. Após avaliar seu aplicativo, o aluno pode acessar a rubrica disponível no próprio sistema para conferir a justificativa das pontuações obtidas e melhorar seu projeto com base nesse *feedback*. De forma similar, um professor pode avaliar os aplicativos dos alunos e receber *feedback* sobre cada projeto individualmente, podendo orientar os alunos conforme o resultado da avaliação.

## 4.3 AUTOMATIZAÇÃO DA AVALIAÇÃO

Para suportar a avaliação e atribuição de nota, automatizou-se a avaliação do design visual baseada na rubrica, evoluindo a ferramenta CodeMaster. O CodeMaster (Gresse von Wangenheim et al., 2018) é um modelo e ferramenta *web* para avaliar automaticamente conceitos de pensamento computacional relacionados a algoritmos e programação em

aplicativos criados com o App Inventor. O CodeMaster foca na avaliação de atividades abertas e complexas de programação sem uma resposta correta única, em que, por exemplo, os alunos desenvolvem aplicativos para resolver problemas relacionados à sua comunidade. Como parte deste trabalho, a ferramenta foi evoluída para incluir a avaliação do design visual de aplicativos criados com o App Inventor por meio de análise estática de código.

Figura 20 – Processo de avaliação automatizada.




Fonte: Elaborada pelo autor

Para avaliar um projeto do App Inventor, o CodeMaster recebe seu código fonte (arquivo .aia exportado do App Inventor) e analisa-o para determinar as propriedades necessárias para aplicar a rubrica (Figura 20). Essa análise é feita inicialmente descompactando o arquivo .aia para acessar as estruturas JSON que descrevem os componentes da IU do aplicativo (botões, caixas de seleção etc.) e suas propriedades (cor, tamanho da fonte etc.). Essas propriedades são então extraídas conforme necessário para determinar a pontuação de cada critério da rubrica e apresentar os resultados da avaliação.

A ferramenta está disponível online em duas línguas, Português Brasileiro e Inglês, no endereço http://apps.computacaonaescola.ufsc.br:8080.

### 4.3.1 Análise de requisitos

Na evolução da ferramenta CodeMaster v1.0, são mantidos os requisitos funcionais originalmente definidos (DEMETRIO, 2017) com alterações nos seguintes requisitos (Tabela 16): Descompactar projeto App Inventor (RF1), Realizar *parse* do projeto App Inventor (RF2), Avaliar projeto App Inventor (RF3), Apresentar avaliação do projeto App Inventor de forma detalhada (RF4) e Apresentar avaliação dos projetos App Inventor de forma resumida (RF5). Os requisitos não-funcionais e casos de uso originalmente definidos no CodeMaster v1.0 (DEMETRIO, 2017) são mantidos.

Tabela 16 – Requisitos elicitados para o CodeMaster UI Design – App Inventor.

| ID | Requisito | Descrição | Artefato de entrada | Artefato de saída |
|---|---|---|---|---|
| RF1 | Descompactar projeto App Inventor | A ferramenta deve ser capaz de descompactar o projeto App Inventor (arquivo .aia) e localizar os arquivos .scm com o design de interface do projeto | Projeto App Inventor (arquivo .aia) | Lista de arquivos .scm de cada tela do aplicativo |
| RF2 | Realizar *parse* do projeto App Inventor | A ferramenta deve realizar o *parse* dos arquivos .scm encontrados no RF1, gerando estruturas de dados JSON correspondentes | Lista de arquivos .scm | Lista de estruturas JSON |
| RF3 | Avaliar projeto App Inventor | A ferramenta deve avaliar cada item da Tabela 14 de acordo com os elementos especificados encontrados no projeto App Inventor e gerar uma nota para cada conceito | Lista de estruturas JSON descrevendo cada tela | Uma resposta de avaliação formada pela nota para cada conceito, um nível de competência e um *feedback* |
| RF4 | Apresentar avaliação do projeto App Inventor de forma detalhada | A ferramenta deve apresentar na interface de usuário a avaliação detalhada do projeto de acordo com a rubrica apresentada na Tabela 14. Deve apresentar a nota de cada conceito separadamente, bem como a nota final e o nível de competência do aluno e o *feedback* | Uma avaliação formada pela nota para cada conceito | Interface de usuário exibe a avaliação (notas de cada conceito, nota final e nível de competência) |
| RF5 | Apresentar avaliação dos projetos App Inventor de forma resumida | A ferramenta deve apresentar na interface de usuário a avaliação de um conjunto de projetos de forma resumida ao professor de acordo com a rubrica apresentada na Tabela 14 | Um conjunto de avaliações, cada uma formada pela nota para cada conceito | Interface de usuário exibe em tabela a nota de cada conceito, nota final e nível para cada avaliação do conjunto, além de uma média de todas as notas finais de cada aluno e o total dos projetos submetidos à análise |

Fonte: Elaborada pelo autor

### 4.3.2 Modelagem e implementação

Para a evolução do CodeMaster v1.0, utiliza-se a linguagem de programação Java com JSP (JavaServer Pages, linguagem Java para desenvolvimento *web*). O módulo de apresentação utiliza as tecnologias JSP, Java Script, HTML5 e CSS3. Essas linguagens e tecnologias são utilizadas por já terem sido utilizadas no desenvolvimento do CodeMaster v1.0.

Para a implementação da rubrica CodeMaster UI Design – App Inventor, são realizadas as seguintes alterações no módulo de apresentação:

- Na tela de professor, é adicionada uma caixa de seleção para cada critério, permitindo escolher quais critérios de design visual devem ser analisados, em uma aba separada das caixas de seleção referentes à rubrica existente de pensamento computacional.
- Nas telas de resultados do aluno e do professor, uma nova aba é adicionada, apresentando os resultados dos critérios de design da mesma forma que os critérios de pensamento computacional, com a diferença de agrupar os critérios por categoria (*Layout*, Tipografia, Escrita, Cores e Imagens).

No módulo de análise e avaliação, são realizadas as seguintes alterações:

- São criadas classes estendendo a classe ConceitoCT, uma para cada critério da rubrica. As novas classes recebem uma instância da classe Código, contendo as estruturas JSON que representam cada tela do aplicativo avaliado.
- A classe AppInventorGrader é evoluída para realizar a avaliação dos conceitos de design de forma análoga à avaliação dos conceitos de pensamento computacional.

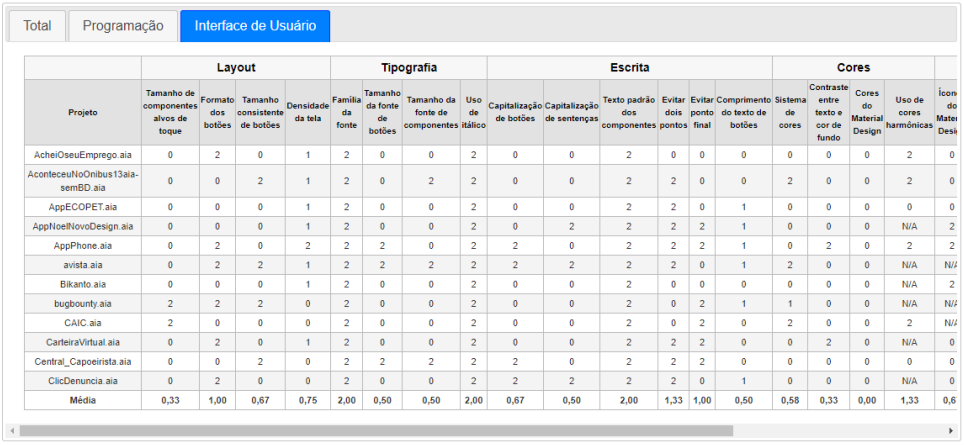
Uma descrição da implementação dos critérios mais elaborados, incluindo pseudocódigos, é apresentada no Apêndice B.

Um novo módulo (Commons) é criado, contendo classes comuns aos dois módulos existentes, incluindo as que anteriormente estavam duplicadas nesses módulos (ProjectSettings e AppInventorGrade). Nesse módulo também é definida uma enumeração dos critérios de design, utilizada na apresentação e avaliação dos critérios, possibilitando verificar automaticamente se todos os critérios listados estão sendo devidamente avaliados. Nas classes ProjectSettings e AppInventorGrade, são adicionadas estruturas de mapa para os critérios de design, evitando a adição de uma variável para cada critério, de forma a minimizar as alterações necessárias para a definição, alteração ou exclusão de critérios.

Figura 21 – Tela de resultados da avaliação realizada por um aluno.


Tela de resultados para o aplicativo apresentado na Figura 19.
Fonte: Elaborada pelo autor

Figura 22 – Tela de resultados da avaliação realizada por um professor.




| | Layout | | | | Tipografia | | | | Escrita | | | | | | Cores | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Projeto | Tamanho de componentes alvos de toque | Formato dos botões | Tamanho consistente de botões | Densidade da tela | Família da fonte | Tamanho da fonte de botões | Tamanho da fonte de componentes | Uso de itálico | Capitalização de botões | Capitalização de sentenças | Texto padrão dos componentes | Evitar dois pontos | Evitar ponto final | Comprimento do texto de botões | Sistema de cores | Contraste entre texto e cor de fundo | Cores do Material Design | Uso de cores harmônicas | Ícon do Mate Desi |
| AcheiOseuEmprego.aia | 0 | 2 | 0 | 1 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| AconteceuNoOnibus13aia-semBD.aia | 0 | 0 | 2 | 1 | 2 | 0 | 2 | 2 | 0 | 0 | 2 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 0 |
| AppECOPET.aia | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| AppNoelNovoDesign.aia | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 2 | 0 | 2 | 2 | 2 | 2 | 1 | 0 | 0 | 0 | N/A | 2 |
| AppPhone.aia | 0 | 2 | 0 | 2 | 2 | 2 | 0 | 2 | 2 | 0 | 2 | 2 | 2 | 1 | 0 | 2 | 0 | 2 | 2 |
| avista.aia | 0 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 | 1 | 2 | 0 | 0 | N/A | N/ |
| Bikanto.aia | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | N/A | 2 |
| bugbounty.aia | 2 | 2 | 2 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 2 | 1 | 1 | 0 | 0 | N/A | N/ |
| CAIC.aia | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 2 | 0 | 0 | 2 | N/ |
| CarteiraVirtual.aia | 0 | 2 | 0 | 1 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 2 | 0 | N/A | 0 |
| Central_Capoeirista.aia | 0 | 0 | 2 | 0 | 2 | 2 | 2 | 2 | 2 | 0 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| ClicDenuncia.aia | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 0 | 1 | 0 | 0 | 0 | N/A | 0 |
| **Média** | 0,33 | 1,00 | 0,67 | 0,75 | 2,00 | 0,50 | 0,50 | 2,00 | 0,67 | 0,50 | 2,00 | 1,33 | 1,00 | 0,50 | 0,58 | 0,33 | 0,00 | 1,33 | 0,6 |

Fonte: Elaborada pelo autor

Como resultado das alterações no módulo de apresentação, o design de interface do CodeMaster UI Design – App Inventor está alinhado ao design de interface do CodeMaster v1.0. Na tela de resultados da avaliação feita por um aluno (requisito RF4 da Tabela 16), os critérios de design são apresentados em uma nova aba (Figura 21). Da mesma forma, os resultados da avaliação de um conjunto de projetos por um professor (requisito RF5 da Tabela 16) também são apresentados em uma nova aba (Figura 22). Para complementar o *feedback* apresentado ao aluno, após avaliar um projeto, ele pode acessar a rubrica por meio do *link* abaixo do nível ninja (Figura 23).

Figura 23 – Tela de visualização da rubrica.

**Rubrica de avaliação App Inventor**

Programação | Interface de Usuário

| Layout |
|---|
| Tipografia |
| Escrita |
| Cores |

| Critério | 0 pontos | 1 ponto | 2 pontos |
|---|---|---|---|
| *Sistema de cores* Quantas cores são usadas no aplicativo (além de preto, branco e cinza)? | 4 ou mais, ou nenhuma | 3 | 1 ou 2 |
| *Contraste entre texto e cor de fundo* Qual é o nível WCAG do aplicativo em relação ao contraste do texto? | Insuficiente | Nível AA | Nível AAA |
| *Cores do Material Design* Usam-se apenas cores da paleta do Material Design? | Não | - | Sim |
| *Uso de cores harmônicas* As tonalidades de cores usadas são harmônicas entre si (complementares, análogas ou triádicas)? | Não | - | Sim |

| Imagens |
|---|

| Critério | 0 pontos | 1 ponto | 2 pontos |
|---|---|---|---|

Fonte: Elaborada pelo autor

### 4.3.3 Testes

A corretude da ferramenta foi avaliada por meio de diversos testes. Os testes realizados incluem a comparação dos resultados da ferramenta com os de avaliação manual e testes de unidade automatizados com aplicativos de teste criados especificamente para esse fim.

Os testes comparando a avaliação automatizada e a manual foram realizados com 8 aplicativos selecionados dentre os mais populares da galeria do App Inventor. Os testes foram realizados pelo autor deste trabalho em conjunto com alunos de graduação em Ciência da Computação (JUSTEN, 2019). Para realizar a análise manual, cada avaliador importou os aplicativos (arquivos .aia) no App Inventor, analisou as propriedades relevantes por meio do Editor de design do App Inventor e determinou assim a pontuação de cada aplicativo em cada critério. Essas pontuações obtidas manualmente foram comparadas com as pontuações resultantes da avaliação automatizada realizada pelo CodeMaster, e, com base nas divergências encontradas, as correções necessárias foram realizadas. Adicionalmente, outros erros de implementação ou problemas na definição dos critérios foram detectados ao analisar outros aplicativos no decorrer das iterações de implementação.

Para os testes de unidade de cada critério, foram criados diversos aplicativos no App Inventor, cobrindo uma variedade de configurações e usos dos componentes que devem ser detectados na avaliação do critério. Para auxiliar nesses testes, as métricas utilizadas para determinar a pontuação de cada critério são registradas pela ferramenta durante a avaliação dos aplicativos. Cada teste de unidade analisa essas métricas, além da pontuação do aplicativo no critério, possibilitando uma depuração mais detalhada em caso de divergências entre a pontuação esperada e a pontuação obtida. Os testes foram escritos utilizando o *framework* de testes JUnit.

# 5 AVALIAÇÃO

Este capítulo apresenta a avaliação da abordagem por meio de um estudo de caso. O objetivo desse estudo é avaliar a rubrica CodeMaster UI Design – App Inventor, que instancia a abordagem, em termos de confiabilidade e validade de construto.

## 5.1 DEFINIÇÃO E COLETA DE DADOS

O objetivo deste estudo é analisar a qualidade da abordagem por meio de um estudo de caso avaliando a rubrica CodeMaster UI Design – App Inventor, que instancia a abordagem. A qualidade é analisada em termos da confiabilidade e validade da rubrica do ponto de vista de pesquisadores no contexto do ensino de design visual como parte do ensino de computação na Educação Básica. A confiabilidade diz respeito à consistência dos itens do modelo, e a validade de um construto indica se ele mede o que realmente pretende medir.

Seguindo a abordagem GQM (BASILI; CALDIERA; ROMBACH, 1994), o objetivo do estudo é definido e decomposto em aspectos de qualidade e perguntas de análise, que são respondidas com base nos dados coletados. Assim, foram definidas as seguintes perguntas de análise, que são respondidas na seção a seguir:

**Confiabilidade**

PA1: Há evidência de consistência interna da rubrica CodeMaster UI Design – App Inventor?

**Validade**

PA2: Há evidência de validade convergente e discriminante da rubrica CodeMaster UI Design – App Inventor?

PA3: Como os fatores subjacentes influenciam as pontuações nos critérios da rubrica CodeMaster UI Design – App Inventor?

PA4: Há evidência de validade das categorias dos itens da rubrica CodeMaster UI Design – App Inventor?

**Coleta de dados.** Foi obtido o código fonte (arquivo .aia) de 88.861 aplicativos da galeria do App Inventor em maio de 2018. O design visual desses aplicativos foi avaliado automaticamente utilizando a ferramenta CodeMaster. Os resultados da avaliação de todos os aplicativos foram exportados como um arquivo CSV para a análise.

Observou-se que grande parte dos aplicativos não possui determinados componentes do design de IU, principalmente imagens (SOLECKI et al., 2020), e, portanto, não recebeu pontuação para alguns critérios. Isso possivelmente se deve ao design da interface de muitos aplicativos estar incompleto, de forma que o conjunto de 88.861 aplicativos não reflete as características do design visual de aplicativos criados no contexto de unidades instrucionais para o ensino de design visual (CNE, 2019; SOLECKI et al., 2020). Portanto, utilizou-se apenas um subconjunto dos 88.861 aplicativos, melhor representando o uso de imagens e outros componentes, e assegurando que cada nível de desempenho dos critérios fosse representado.

Tabela 17 – Frequência de aplicativos por nível de desempenho.

| **Critério** | **Frequência (%) das pontuações** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Total (88.861 aplicativos)** | | | | **Amostra (1.870 aplicativos)** | | | |
| | **0pt** | **1pt** | **2pt** | **NA** | **0pt** | **1pt** | **2pt** | **NA** |
| L1 tamanhoDeAlvosDeToque | 32 | - | 65 | 3 | 44 | - | 56 | 0 |
| L2 formatoDosBotoes | 11 | - | 64 | 24 | 19 | - | 59 | 23 |
| L3 tamanhoConsistenteDeBotoes | 46 | - | 41 | 13 | 56 | - | 44 | 0 |
| L4 densidadeDaIU | 30 | 15 | 55 | 0 | 31 | 16 | 53 | 0 |
| T1 familiaFonte | 11 | - | 78 | 12 | 18 | - | 82 | 0 |
| T2 tamanhoFonteBotoes | 25 | - | 53 | 22 | 35 | - | 44 | 21 |
| T3 tamanhoDaFonte | 29 | - | 53 | 18 | 43 | - | 45 | 13 |
| T4 naoUsaItalico | 12 | - | 79 | 9 | 20 | - | 80 | 0 |
| E1 caixaAltaBotao | 68 | - | 9 | 22 | 64 | - | 15 | 21 |
| E2 capitalizacaoSentenca | 28 | - | 72 | 0 | 35 | - | 65 | 0 |
| E3 textoDefault | 12 | - | 75 | 13 | 16 | - | 84 | 0 |
| E4 doisPontosLegenda | 15 | - | 46 | 40 | 18 | - | 53 | 29 |
| E5 pontoFinalSentenca | 13 | - | 87 | 0 | 19 | - | 81 | 0 |
| E6 tamanhoMaximoTextoBotao | 30 | 25 | 23 | 22 | 34 | 22 | 23 | 21 |
| C1 sistemaDeCores | 67 | 10 | 23 | 0 | 65 | 14 | 21 | 0 |
| C2 contrasteEntreTextoFundo | 28 | 7 | 53 | 12 | 37 | 14 | 49 | 0 |
| C3 coresMaterialDesign | 98 | - | 2 | 0 | 87 | - | 13 | 0 |
| C4 coresHarmonicas | 22 | - | 31 | 47 | 28 | - | 40 | 33 |
| I1 iconesMD | 20 | - | 0 | 80 | 54 | - | 1 | 44 |
| I2 pixelizacao | 23 | - | 28 | 48 | 37 | - | 31 | 32 |
| I3 distorcao | 33 | - | 20 | 46 | 49 | - | 19 | 31 |

Fonte: Elaborada pelo autor

Para compor a amostra, foram incluídos pelo menos 250 aplicativos aleatórios para cada um dos 2 ou 3 níveis de desempenho de cada critério e, ao todo, pelo menos 750 com pontuação (ou seja, não NA) em cada critério, com preferência por aplicativos com NA em 6 critérios (mediana dos 88.861 aplicativos) para evitar que a amostra continuasse com excesso de NAs. Dessa forma, utilizou-se uma amostra mais equilibrada de 1870 aplicativos (Tabela 17). Essa amostra apresenta, em relação ao conjunto de 88.861 aplicativos, porcentagens em

geral menores de aplicativos com NA, especialmente nos critérios relacionados a imagens. Além disso, ela apresenta uma distribuição mais balanceada entre os níveis de desempenho dos critérios em geral, destacando-se o critério coresMaterialDesign. O critério íconesMD, no entanto, ainda apresenta uma porcentagem muito baixa (1%) de aplicativos com pontuação 2pt na amostra.

## 5.2 ANÁLISE

As duas principais propriedades psicométricas da abordagem, a confiabilidade e a validade, são avaliadas com base na Teoria Clássica dos Testes (TCT) (PASQUALI, 1997). A confiabilidade está relacionada à reprodutibilidade da medida, e é avaliada utilizando o coeficiente alfa de Cronbach (CRONBACH, 1951). A validade se refere à capacidade da abordagem de medir um conceito teórico específico, que, neste caso, é a qualidade do design visual. Para determinar a validade, utilizam-se as correlações (DEVELLIS, 2003) e a análise fatorial (GLORFELD, 1995).

Além da análise segundo a TCT, é realizada uma análise pela Teoria da Resposta ao Item (TRI) (ANDRADE; TAVARES; VALLE, 2000), a teoria da psicometria moderna. A TRI difere da TCT pelo uso de escalas mais curtas, conservando as propriedades psicométricas do instrumento. A TRI assume a existência de um traço latente (neste caso, a qualidade do design visual). Para a análise via TRI, utiliza-se o Modelo de Resposta Gradual (MRG) (SAMEJIMA, 1969).

Para realizar as análises, foi utilizada a linguagem de programação R. Cada pergunta de análise é respondida a seguir com base nos resultados das análises.

### 5.2.1 Há evidência de consistência interna da rubrica CodeMaster UI Design – App Inventor?

A confiabilidade da rubrica CodeMaster UI Design – App Inventor foi analisada medindo sua consistência interna por meio do coeficiente alfa de Cronbach (CRONBACH, 1951). Esse coeficiente indica indiretamente o grau em que um conjunto de critérios mede um único fator de qualidade. No caso deste estudo, espera-se que todos os critérios da rubrica meçam o mesmo fator: a conformidade do design de interface com diretrizes de design visual.

Em geral, valores de alfa de Cronbach a partir de 0,70 são considerados aceitáveis, indicando consistência interna (DEVELLIS, 2003). Analisando os 21 critérios da rubrica, obteve-se um valor satisfatório do coeficiente alfa de Cronbach ($\alpha = 0,84$).

Complementando esta análise, o coeficiente alfa de Cronbach foi calculado para cada uma das categorias (Tabela 18). Os valores obtidos são inferiores ao da rubrica integral. No entanto, isso indica que a totalidade dos critérios da rubrica em conjunto contribuem de fato para a avaliação da qualidade do design de interface, mais do que cada categoria individualmente. Comparando o valor do coeficiente entre as categorias, a análise indica uma menor consistência nas categorias Escrita e Imagens, que obtiveram os menores valores para o coeficiente, o que é confirmado e discutido nas análises a seguir.

Tabela 18 – Alfa de Cronbach por categoria.

| | ***Layout*** | **Tipografia** | **Escrita** | **Cores** | **Imagens** |
|---|---|---|---|---|---|
| **Alfa (α)** | 0,61 | 0,66 | 0,58 | 0,65 | 0,51 |

Fonte: Elaborada pelo autor

### 5.2.2 Há evidência de validade convergente e discriminante da rubrica CodeMaster UI Design – App Inventor?

Para determinar se as dimensões da rubrica foram definidas de forma adequada, foram analisadas as validades convergente e discriminante dos critérios da rubrica. Para isso, foram calculadas as intercorrelações entre critérios e a correlação critério-total (WOHLIN et al., 2012).

**Intercorrelações entre critérios.** Para determinar a validade convergente, espera-se que os critérios da mesma dimensão (tipografia, cor etc.) tenham uma alta correlação (TROCHIM; DONNELLY, 2008; CARMINES; ZELLER, 1982). Por outro lado, para determinar a validade discriminante, espera-se que critérios de diferentes dimensões tenham baixa correlação, visto que teoricamente medem diferentes dimensões.

Para analisar as intercorrelações entre critérios de uma mesma dimensão, foi utilizada a correlação policórica (Tabela 19), a mais apropriada para variáveis ordinais observadas (OLSSON, 1979). A matriz apresenta os coeficientes de correlação, indicando o grau de correlação entre cada par de critérios. O coeficiente para cada par de critérios é calculado com base nas pontuações que todos os aplicativos obtiveram nesses critérios. Os coeficientes de correlação entre critérios da mesma dimensão estão coloridos na Tabela 19.

Tabela 19 – Resultados da análise de correlação policórica.

| | | Layout | | | | Tipografia | | | | Escrita | | | | | | Cores | | | | Imagens | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | L1 | L2 | L3 | L4 | T1 | T2 | T3 | T4 | W1 | W2 | W3 | W4 | W5 | W6 | C1 | C2 | C3 | C4 | I1 | I2 | I3 |
| Layout | L1 tamanhoDeAlvosDeToque | 1.00 | | | | | | | | | | | | | | | | | | | | |
| | L2 formatoDosBotoes | **0.47** | 1.00 | | | | | | | | | | | | | | | | | | | |
| | L3 tamanhoConsistenteDeBoto | **0.36** | **0.54** | 1.00 | | | | | | | | | | | | | | | | | | |
| | L4 densidadeDaIU | **0.44** | **0.47** | **0.45** | 1.00 | | | | | | | | | | | | | | | | | |
| Tipografia | T1 familiaFonte | **0.31** | **0.46** | 0.29 | **0.36** | 1.00 | | | | | | | | | | | | | | | | |
| | T2 tamanhoFonteBotoes | **0.38** | **0.54** | **0.30** | **0.39** | **0.52** | 1.00 | | | | | | | | | | | | | | | |
| | T3 tamanhoDaFonte | 0.24 | **0.56** | **0.31** | **0.45** | **0.54** | **0.66** | 1.00 | | | | | | | | | | | | | | |
| | T4 evitarItalico | **0.33** | **0.48** | **0.32** | **0.43** | **0.51** | **0.45** | **0.42** | 1.00 | | | | | | | | | | | | | |
| Escrita | E1 caixaAltaBotao | 0.17 | 0.21 | 0.22 | 0.18 | 0.11 | 0.05 | 0.21 | 0.11 | 1.00 | | | | | | | | | | | | |
| | E2 capitalizacaoSentenca | 0.27 | **0.32** | **0.34** | **0.37** | 0.26 | 0.18 | 0.26 | **0.29** | **0.43** | 1.00 | | | | | | | | | | | |
| | E3 textoPadrão | 0.24 | 0.20 | **0.31** | **0.40** | 0.14 | 0.06 | 0.16 | 0.13 | **0.37** | 0.20 | 1.00 | | | | | | | | | | |
| | E4 doisPontosLegenda | **0.31** | 0.22 | **0.35** | **0.42** | **0.34** | 0.26 | **0.32** | 0.28 | 0.27 | **0.35** | 0.13 | 1.00 | | | | | | | | | |
| | E5 pontoFinalSentenca | **0.46** | **0.46** | **0.47** | **0.51** | **0.47** | **0.39** | **0.54** | **0.44** | 0.23 | **0.36** | 0.26 | **0.37** | 1.00 | | | | | | | | |
| | E6 tamanhoMaximoTextoBotao | 0.24 | **0.50** | **0.45** | **0.44** | **0.30** | **0.35** | **0.46** | **0.39** | **0.40** | 0.21 | **0.58** | 0.19 | **0.42** | 1.00 | | | | | | | |
| Cores | C1 sistemaDeCores | **0.39** | **0.43** | **0.44** | **0.37** | **0.29** | **0.32** | **0.33** | **0.31** | -0.02 | 0.16 | 0.20 | 0.18 | **0.42** | 0.26 | 1.00 | | | | | | |
| | C2 contrasteEntreTextoFundo | **0.46** | **0.50** | **0.48** | **0.40** | **0.43** | **0.45** | **0.37** | **0.44** | 0.25 | **0.33** | 0.20 | 0.26 | **0.50** | **0.38** | **0.45** | 1.00 | | | | | |
| | C3 coresMaterialDesign | 0.23 | **0.35** | **0.42** | **0.33** | 0.22 | 0.14 | 0.18 | 0.23 | -0.10 | 0.11 | 0.12 | 0.19 | 0.29 | 0.15 | **0.61** | **0.42** | 1.00 | | | | |
| | C4 coresHarmonicas | 0.25 | **0.37** | **0.51** | **0.30** | 0.25 | 0.26 | 0.29 | 0.27 | -0.01 | 0.24 | 0.13 | **0.30** | **0.37** | 0.24 | **0.42** | **0.47** | **0.60** | 1.00 | | | |
| Imagens | I1 iconesMD | -0.08 | 0.01 | 0.00 | -0.11 | 0.18 | 0.16 | 0.13 | 0.12 | 0.09 | 0.08 | -0.02 | 0.19 | 0.12 | 0.13 | -0.59 | 0.01 | -0.60 | 0.10 | 1.00 | | |
| | I2 pixelizacao | **0.47** | 0.22 | 0.25 | **0.33** | 0.28 | **0.37** | 0.28 | 0.25 | 0.04 | 0.17 | 0.23 | 0.24 | 0.25 | **0.30** | 0.23 | 0.23 | 0.09 | 0.27 | 0.20 | 1.00 | |
| | I3 distorcao | **0.64** | **0.47** | **0.39** | **0.35** | **0.43** | **0.51** | **0.36** | **0.45** | 0.10 | **0.31** | 0.28 | **0.51** | **0.49** | **0.38** | **0.48** | **0.46** | 0.08 | **0.48** | **0.29** | **0.72** | 1.00 |

Fonte: Elaborada pelo autor

De acordo com Cohen (1998), uma correlação entre os critérios é considerada satisfatória se o coeficiente de correlação for maior que 0,29, indicando que há uma correlação média ou alta entre os critérios. As correlações com valor satisfatório estão marcadas em negrito na Tabela 19.

Analisando a correlação entre pares de critérios da mesma dimensão, todos os pares nas dimensões *Layout*, Tipografia e Cores apresentam um grau aceitável de correlação, com valores variando de 0,36 a 0,66. Vários pares da dimensão Escrita apresentam valor abaixo de 0,29. Essas baixas correlações podem ser associadas ao fato de que as recomendações do Material Design quanto à escrita diferem do recomendado em outros contextos. Por exemplo, escrever texto completamente em maiúsculo, como recomendado para botões, geralmente é considerado inapropriado (NIELSEN, 2005), assim como a omissão do ponto ao final de sentenças difere das regras usuais de escrita. Assim, pode ser necessário reformular os critérios da categoria, embora sejam derivados das diretrizes do Material Design, ou utilizar uma amostra que represente o aprendizado dessas regras não usuais de escrita.

A dimensão Imagens também apresenta um par com correlação abaixo de 0,29 (I1–I2), o que se associa a problemas identificados no item I1 (ÍconesMD), indicados nas análises a seguir. Nenhum valor negativo foi encontrado entre pares da mesma dimensão.

Quanto à correlação entre critérios de diferentes dimensões, vários pares de critérios também apresentam uma alta correlação. Em particular, isso ocorre com a maioria dos pares de critérios entre as dimensões *Layout* e Tipografia. Consequentemente, não é possível estabelecer

a validade discriminante. No entanto, isso não é considerado um problema, visto que todos os critérios estão relacionados ao mesmo fator, a qualidade do design visual.

**Correlação critério-total.** Complementando a análise anterior, avaliou-se também a correlação entre cada critério e todos os demais. Cada critério do instrumento deve ter uma correlação média ou alta com todos os outros critérios (DEVELLIS, 2003), pois isso indica que os critérios são consistentes comparados com os outros critérios. Um critério com baixa correlação critério-total enfraquece a validade da rubrica, indicando que ele deve ser revisado ou eliminado.

Tabela 20 – Correlação item-total e alfa de Cronbach com itens removidos.

| Item | Correlação item-total | Alfa de Cronbach removendo o item |
|---|---|---|
| L1 tamanhoDeAlvosDeToque | 0,46 | 0,83 |
| L2 formatoDosBotoes | 0,50 | 0,82 |
| L3 tamanhoConsistenteDeBotoes | 0,47 | 0,83 |
| L4 densidadeDaIU | 0,53 | 0,82 |
| T1 familiaFonte | 0,41 | 0,83 |
| T2 tamanhoFonteBotoes | 0,45 | 0,83 |
| T3 tamanhoDaFonte | 0,46 | 0,83 |
| T4 evitarItalico | 0,42 | 0,83 |
| E1 caixaAltaBotao | **0,17** | **0,84** |
| E2 capitalizacaoSentenca | 0,34 | 0,83 |
| E3 textoPadrao | **0,24** | 0,83 |
| E4 doisPontosLegenda | 0,34 | 0,83 |
| E5 pontoFinalSentenca | 0,50 | 0,82 |
| E6 tamanhoMaximoTextoBotao | 0,46 | 0,83 |
| C1 sistemaDeCores | 0,43 | 0,83 |
| C2 contrasteEntreTextoFundo | 0,55 | 0,82 |
| C3 coresMaterialDesign | **0,25** | 0,83 |
| C4 coresHarmonicas | 0,41 | 0,83 |
| I1 iconesMD | **0,05** | **0,84** |
| I2 pixelizacao | 0,34 | 0,83 |
| I3 distorcao | 0,48 | 0,82 |

**Negrito** – Valor insatisfatório (baixa correlação/aumento do alfa)

Fonte: Elaborada pelo autor

Para essa análise, utilizou-se o método da correlação critério-total corrigida, que compara um critério com todos os outros da rubrica. Os valores de referência para a análise são os mesmos da análise anterior, seguindo Cohen (1998), considerando uma correlação satisfatória se o coeficiente de correlação for maior que 0,29. Além disso, analisou-se o alfa de Cronbach quando cada um dos critérios é excluído, esperando que nenhum critério cause um aumento no alfa de Cronbach quando excluído (WOHLIN et al., 2012). Os resultados desta análise são apresentados na Tabela 20.

O valor do alfa de Cronbach sofre um aumento ao excluir os itens E1 (caixaAltaBotão) e I1 (íconesMD). A exclusão dos demais itens não causa aumento no alfa, indicando que eles contribuem para a avaliação. Quanto à correlação item-total, a maioria dos itens apresenta uma correlação acima de 0,29. No entanto, 4 itens apresentam uma correlação baixa (E1, E3, C3, I1).

Entre os itens com baixa correlação, alguns itens (E1 – caixaAltaBotão, C3 – coresMD, I1 – íconesMD) estão mais diretamente relacionados com padrões estabelecidos pelo Material Design do que propriamente os princípios de design visual. Assim, visto que os aplicativos provêm da galeria do App Inventor, a dificuldade com esses itens pode estar relacionada ao desconhecimento das diretrizes do Material Design, de forma que os aplicativos que melhor aplicam os princípios de design visual não necessariamente seguem essas recomendações mais específicas do Material Design.

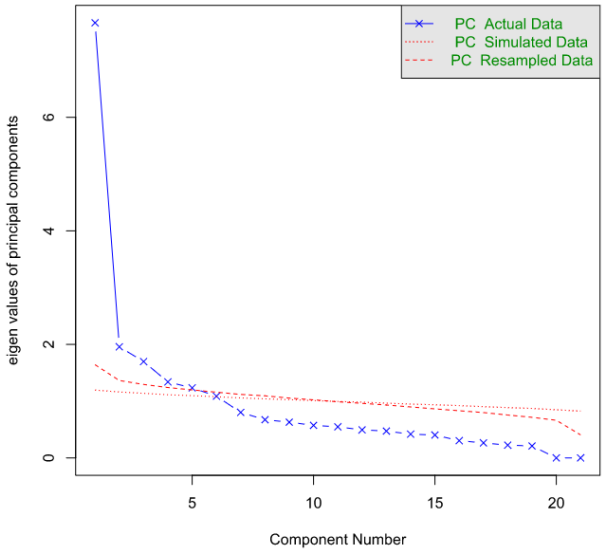
O item I1 (íconesMD) especificamente não está bem representado na amostra, mesmo utilizando uma amostra mais equilibrada, de forma que muito poucos aplicativos pontuaram no critério (ver Tabela 17). Esse desbalanceamento pode ser atribuído à origem dos aplicativos, pois entende-se que os aplicativos da galeria não representam bem o aprendizado desse conceito. Isso se agrava devido à forma como o critério foi definido, considerando todas as imagens em botões como ícones, o que tende a aumentar a quantidade de aplicativos que não pontuam no critério. O uso de uma definição mais precisa não foi possível devido a uma dificuldade na implementação (conforme indicado na Tabela 13).

### 5.2.3 Como os fatores subjacentes influenciam as pontuações nos itens da rubrica CodeMaster UI Design – App Inventor?

Para identificar quantos fatores representam a aplicação dos conceitos avaliados pela rubrica CodeMaster UI Design – App Inventor, realizou-se uma análise fatorial. Para determinar se os critérios da rubrica poderiam ser submetidos a uma análise fatorial, utilizou-se o índice Kaiser-Meyer-Olkin (KMO) (BROWN, 2006). O índice KMO indica quão adequada é a amostragem com um valor entre 0,0 e 1,0. Um valor próximo a 1,0 indica que a amostra suporta uma análise fatorial, enquanto valores menores que 0,5 indicam que a análise fatorial provavelmente não é aplicável (BROWN, 2006). O índice KMO obtido para o conjunto de critérios da rubrica CodeMaster UI Design – App Inventor foi de 0,84, indicando que a análise fatorial é apropriada.

A análise fatorial é utilizada para definir o número de fatores a serem retidos (GLORFELD, 1995). Foi utilizado o método da análise paralela, que indicou a existência de um fator predominante (Figura 24). No entanto, observam-se mais dois pontos que se destacam acima da linha vermelha.

Figura 24 – *Scree plot* referente à rubrica CodeMaster UI Design – App Inventor.




Fonte: Elaborada pelo autor

Tabela 21 – Cargas fatoriais para 3 fatores na rubrica CodeMaster UI Design – App Inventor.

| Item | Fator 1 | Fator 2 | Fator 3 |
|---|---|---|---|
| tamanhoDeAlvosDeToque | **-0,424** | **0,188** | **-0,129** |
| formatoDosBotoes | **-0,474** | **0,201** | **-0,241** |
| tamanhoConsistenteDeBotoes | **-0,082** | 0,484 | **-0,370** |
| densidadeDaIU | **-0,323** | **0,145** | **-0,402** |
| familiaFonte | **-0,684** | **0,016** | **-0,019** |
| tamanhoFonteBotoes | **-0,741** | **-0,053** | **-0,007** |
| tamanhoDaFonte | **-0,622** | **0,009** | **-0,077** |
| evitarItalico | **-0,538** | **0,096** | **-0,113** |
| caixaAltaBotao | **0,020** | **-0,164** | **-0,662** |
| capitalizacaoSentenca | **-0,165** | **0,104** | **-0,316** |
| textoPadrao | **0,108** | **-0,006** | **-0,794** |
| doisPontosLegenda | **-0,349** | **0,051** | **-0,180** |
| pontoFinalSentenca | **-0,440** | **0,249** | **-0,228** |
| tamanhoMaximoTextoBotao | **-0,205** | **-0,013** | **-0,688** |
| sistemaDeCores | **-0,250** | *0,642* | **0,008** |
| contrasteEntreTextoFundo | **-0,296** | 0,449 | **-0,135** |
| coresMaterialDesign | **0,098** | *0,916* | **0,046** |
| coresHarmonicas | **-0,042** | *0,772* | **0,066** |
| iconesMD | **-0,627** | **-0,211** | **0,304** |
| pixelizacao | **-0,632** | **-0,091** | **-0,073** |
| distorcao | **-0,809** | **0,131** | **0,040** |

*Itálico* – Valor acima de 0,55 (bom)
**Negrito** – Valor abaixo de 0,45 (ruim)

Fonte: Elaborada pelo autor

Observando que há 3 pontos acima da linha vermelha, o *scree plot* sugere que pode haver 3 fatores subjacentes. Para determinar quais itens são carregados em qual fator, utiliza-se o método de rotação Oblimin, no qual os fatores podem ser correlacionados (JACKSON, 2005). As cargas fatoriais dos itens associados aos 3 fatores retidos são apresentadas na Tabela 21.

Os limites para classificar as cargas fatoriais, conforme definidos por Comrey e Lee (1992), são: 0,32 (ruim), 0,45 (aceitável), 0,55 (bom), 0,63 (muito bom) ou 0,71 (excelente). Analisando as cargas fatoriais na Tabela 21, observa-se que dois fatores apresentam carga fatorial baixa ou negativa para todos os itens, o que indica de fato a existência de apenas um fator, a qualidade do design visual.

As cargas calculadas para um único fator são apresentadas na Tabela 22. Nota-se que dois terços dos itens apresentam valor acima do limiar 0,55 (bom). Apenas 4 itens estão abaixo do valor 0,45 (aceitável), marcados em negrito na Tabela 22: caixaAltaBotão, capitalizacaoSentenca, textoPadrão e íconesMD.

Tabela 22 – Carregamento em um fator para a rubrica CodeMaster UI Design – App Inventor.

| Item | Fator |
|---|---|
| tamanhoDeAlvosDeToque | *0,588* |
| formatoDosBotoes | *0,708* |
| tamanhoConsistenteDeBotoes | *0,648* |
| densidadeDaIU | *0,635* |
| familiaFonte | *0,604* |
| tamanhoFonteBotoes | *0,580* |
| TamanhoDaFonte | *0,567* |
| evitarItalico | *0,600* |
| caixaAltaBotão | **0,261** |
| capitalizacaoSentenca | **0,416** |
| textoPadrao | **0,385** |
| doisPontosLegenda | 0,455 |
| pontoFinalSentenca | *0,710* |
| tamanhoMaximoTextoBotao | 0,538 |
| sistemaDeCores | *0,679* |
| contrasteEntreTextoFundo | *0,675* |
| coresMaterialDesign | *0,592* |
| coresHarmonicas | 0,543 |
| íconesMD | **0,195** |
| Pixelização | 0,501 |
| Distorção | *0,764* |

*Itálico* – Valor acima de 0,55 (bom)
**Negrito** – Valor abaixo de 0,45 (ruim)

Fonte: Elaborada pelo autor

### 5.2.4 Existe evidência de validade das categorias dos itens da rubrica CodeMaster v2.0?

A validade das categorias dos itens é analisada por meio da Teoria da Resposta ao Item (TRI) (ANDRADE; TAVARES; VALLE, 2000), utilizando o Modelo de Resposta Gradual (MRG) de Samejima (1969), que assume que as categorias de resposta de um item são ordenadas entre si (SAMEJIMA, 1969). Os parâmetros populacionais μ (média) e σ (desvio-padrão), referentes às habilidades da população considerada, são fixados como μ = 0 e σ = 1. O MRG determina para cada item:

a) o parâmetro de inclinação (*a*);
b) a distância ($b_1$ e $b_2$) entre as categorias de resposta (ou seja, entre 0pt–1pt, e entre 1pt–2pt, para itens com as 3 categorias de resposta).

Ao estimar (ou "calibrar") os parâmetros, valores do parâmetro de inclinação (*a*) são considerados bons se forem maiores do que 1,0. Para os parâmetros de dificuldade $b_1$ e $b_2$, espera-se em geral que os valores estejam entre -5,0 e 5,0.

Tabela 23 – Parâmetros da TRI para os itens da rubrica CodeMaster UI Design – App Inventor.

| Item | a | EP(a) | b1 | EP(b1) | b2 | EP(b2) |
|---|---|---|---|---|---|---|
| L1 tamanhoDeAlvosDeToque | 1,238 | 0,080 | -0,251 | 0,051 | | |
| L2 formatoDosBotoes | 1,708 | 0,125 | -1,168 | 0,062 | | |
| L3 tamanhoConsistenteDeBotoes | 1,449 | 0,093 | 0,254 | 0,045 | | |
| L4 densidadeDaIU | 1,401 | 0,079 | -0,797 | 0,055 | -0,127 | 0,045 |
| T1 familiaFonte | 1,290 | 0,094 | -1,558 | 0,092 | | |
| T2 tamanhoFonteBotoes | 1,214 | 0,086 | -0,346 | 0,056 | | |
| T3 tamanhoDaFonte | 1,172 | 0,081 | -0,115 | 0,054 | | |
| T4 evitarItalico | 1,276 | 0,090 | -1,399 | 0,084 | | |
| E1 caixaAltaBotao | 0,461 | 0,078 | 3,195 | 0,533 | | |
| E2 capitalizacaoSentenca | 0,780 | 0,064 | -0,899 | 0,092 | | |
| E3 textoPadrao | 0,710 | 0,077 | -2,617 | 0,255 | | |
| E4 doisPontosLegenda | 0,869 | 0,085 | -1,699 | 0,136 | | |
| E5 pontoFinalSentenca | 1,718 | 0,115 | -1,249 | 0,064 | | |
| E6 tamanhoMaximoTextoBotao | 1,086 | 0,072 | -0,472 | 0,062 | 0,897 | 0,080 |
| C1 sistemaDeCores | 1,575 | 0,105 | 0,584 | 0,047 | 1,197 | 0,066 |
| C2 contrasteEntreTextoFundo | 1,556 | 0,087 | -0,465 | 0,047 | 0,062 | 0,043 |
| C3 coresMaterialDesign | 1,251 | 0,122 | 1,881 | 0,136 | | |
| C4 coresHarmonicas | 1,102 | 0,088 | -0,650 | 0,067 | | |
| I1 iconesMD | 0,338 | 0,225 | 10,796 | 7,286 | | |
| I2 pixelizacao | 0,985 | 0,085 | -0,038 | 0,069 | | |
| I3 distorcao | 2,017 | 0,166 | 0,528 | 0,053 | | |

Fonte: Elaborada pelo autor

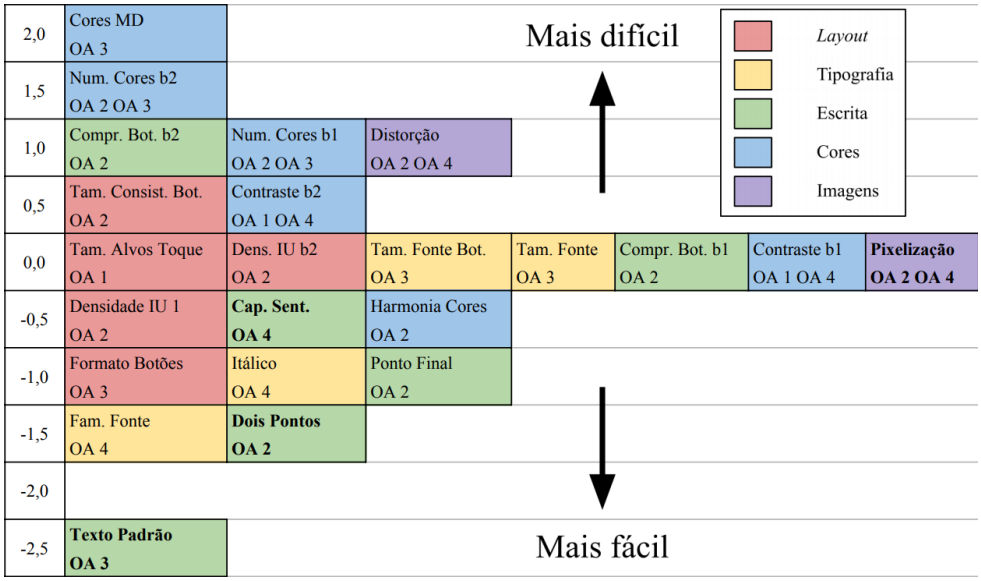
Como resultado da calibração utilizando a avaliação de 1870 aplicativos da galeria do App Inventor realizada pelo CodeMaster UI Design – App Inventor, a maioria dos itens atingiu uma boa calibração, com parâmetro de inclinação (*a*) maior que 1,0 (Tabela 23). No entanto, alguns itens apresentaram um parâmetro de inclinação baixo, em maioria, na categoria Escrita.

O item íconesMD apresenta o valor mais baixo de parâmetro de inclinação ($a$ = 0,338), bem como valores de erro padrão maiores do que os demais itens. Todos os itens possuem parâmetros de dificuldade ($b_1$ e $b_2$) entre -3,0 e 3,5, com exceção do item íconesMD, que apresentou um valor muito alto ($b_1$ = 10,796).

Diante desses resultados, a análise via TRI confirma os resultados da análise das correlações. Os valores do item ÍconesMD podem ser explicados, conforme indicado na análise da correlação critério-total (Seção 5.2.2), pela forma como o critério foi definido, além de a amostra não representar bem o aprendizado do critério. Quanto aos itens da categoria Escrita, presume-se que os valores baixos de parâmetro de inclinação se devem ao conjunto de dados, que também não representa bem o aprendizado desses conceitos, sendo necessário o uso de uma amostra que represente melhor o aprendizado das diretrizes de escrita do Material Design.

**Posicionamento na escala.** Todos os itens considerados bem calibrados, com parâmetro de inclinação ($a$) acima de 1,0, são posicionados numa escala com μ = 0 e σ = 1 (Figura 25). Itens com parâmetro de inclinação entre 0,7 e 1,0 não atingiram uma boa calibração, e esse valor baixo indica que o item tem pouco poder de discriminação, não diferenciando devidamente indivíduos com diferentes níveis de habilidade. Assim, esses itens foram incluídos na escala destacados em negrito, indicando esse problema na calibração, que pode ser consequência de os itens terem sido definidos de forma inadequada ou de a amostra não representar bem os diferentes níveis de habilidade.

Figura 25 – Posicionamento dos itens na escala.




**Negrito** – Item que não calibrou bem (0,7 < a < 1,0)

Fonte: Elaborada pelo autor

Nessa escala de habilidade, a relação de ordem entre os pontos é o importante, e não os valores absolutos dos pontos. Os itens foram posicionados nos pontos da escala de acordo com os parâmetros de dificuldade $b_1$ e $b_2$ calibrados (apresentados na Tabela 23). O item I3 – distorção, por exemplo, possui parâmetro de dificuldade calibrado $b_1$ = 0,528, de modo que Distorção b1 é posicionado no ponto 1,0 (o ponto mais próximo de valor maior do que $b_1$).

Cada um dos parâmetros $b_1$ e $b_2$ representa a habilidade necessária para que um indivíduo tenha probabilidade de acerto de 50%. Assim, com base na escala (0,1), um indivíduo com habilidade 2,0 (2 desvios-padrão acima da habilidade média) tem probabilidade maior do que 50% de acertar os itens posicionados abaixo do ponto 2,0.

A posição dos itens na escala sugere um sequenciamento baseado na dificuldade para o conteúdo de uma unidade instrucional para o ensino de design visual. Atualmente, não há uma clara definição de como deve ser esse sequenciamento, visto que as unidades instrucionais existentes não abordam o design visual de forma detalhada (FERREIRA et al., 2019a).

Analisando os critérios por categoria ou por objetivo de aprendizagem (conforme indicado na Figura 25), observa-se de forma geral que todas as categorias e objetivos de aprendizagem possuem critérios espalhados dos níveis baixos aos níveis altos da escala. Assim, nenhum objetivo de aprendizagem ou categoria se mostrou mais fácil ou mais difícil que os demais, sugerindo uma estratégia de ensino abordando todos os elementos do design visual de forma gradual, definindo a sequência conforme a dificuldade de cada item individualmente.

### 5.2.5 Discussão

De forma geral, os resultados indicam que a rubrica CodeMaster UI Design – App Inventor de avaliação do design visual apresenta boa confiabilidade e validade. Alguns itens, principalmente na categoria Escrita, apresentaram resultados inadequados, indicando que podem não estar realmente contribuindo para a avaliação. No entanto, entende-se que isso se deve à origem da amostra, visto que os aplicativos provêm da galeria do App Inventor, e não se conhece o contexto de desenvolvimento desses aplicativos pois a galeria não contém essa informação. Assim, os aplicativos incluídos na amostra não representam bem o aprendizado de design visual, pois não foram necessariamente desenvolvidos em contexto educacional e, mesmo que tenham sido, a maioria das unidades instrucionais para o ensino de computação atualmente não aborda conceitos de design visual. Portanto, se a amostra for coletada como resultado de unidades instrucionais que abordam o design de IU, os resultados da análise podem mudar, especialmente para critérios que representam diretrizes específicas do Material Design.

A análise fatorial revela que a maioria dos itens apresenta alto carregamento em um fator. Embora os itens estejam organizados em diversas categorias, a análise paralela não indica a existência de fatores relacionados a cada categoria. Assim, identifica-se apenas um fator, a qualidade do design visual.

Como resultado da análise via TRI, a maioria dos itens apresentou bons parâmetros de inclinação. Apenas alguns itens, principalmente na categoria Escrita, apresentaram valores abaixo do esperado, ou seja, não calibraram bem, o que indica que não diferenciam de forma apropriada diferentes níveis de habilidade. Entende-se que esses resultados foram obtidos porque a amostra não representa bem o aprendizado dessas diretrizes. O item ÍconesMD, em particular, apresentou os piores resultados na calibração, pois a amostra contém uma quantidade insuficiente de aplicativos que pontuam nesse item. No entanto, acredita-se que isso pode ser solucionado utilizando uma amostra de aplicativos desenvolvidos como resultado de unidades instrucionais envolvendo o design de IU, resultando em uma melhor calibração.

### 5.2.6 Ameaças à validade

A fim de mitigar as ameaças relacionadas ao design do estudo, foi definida e documentada uma metodologia sistemática usando a abordagem GQM (BASILI; CALDIERA; ROMBACH, 1994). Um risco está relacionado à qualidade dos dados em termos de padronização de dados. Como este estudo é limitado exclusivamente a avaliações usando a rubrica CodeMaster UI Design – App Inventor, esse risco é mitigado, pois todas as avaliações foram realizadas de maneira automatizada usando a mesma rubrica. Outra questão refere-se ao agrupamento de dados de diferentes contextos. Os aplicativos do conjunto de dados vêm de diversos contextos da comunidade do App Inventor em todo o mundo, e nenhuma informação adicional sobre o histórico dos criadores dos programas na Galeria do App Inventor está disponível. No entanto, esse conjunto de dados possibilita analisar a validade da rubrica de maneira independente do contexto, minimizando o impacto da escolha desse conjunto.

Em termos de validade externa, uma ameaça à possibilidade de generalizar os resultados está relacionada ao tamanho da amostra e à diversidade de dados utilizados para a avaliação. A análise é baseada em projetos coletados da Galeria do App Inventor, envolvendo uma amostra de 1,870 aplicativos da comunidade do App Inventor em todo o mundo. Este é considerado um tamanho de amostra satisfatório, permitindo a geração de resultados significativos. Em termos de confiabilidade, uma ameaça refere-se a até que ponto os dados e

a análise dependem de pesquisadores específicos. A fim de mitigar essa ameaça, adotou-se uma metodologia sistemática, definindo claramente o objetivo do estudo, o processo de coleta de dados e os métodos estatísticos utilizados para a análise de dados. Outra questão refere-se à escolha correta dos métodos estatísticos para análise de dados. Para minimizar essa ameaça, foi realizada uma análise estatística baseada na abordagem para a construção de escalas de medida proposta por DeVellis (2003), que está alinhada com procedimentos para a avaliação da consistência interna e validade de construto de instrumentos de medição.

# 6 CONCLUSÃO

Como resultado deste trabalho, foi desenvolvida uma abordagem confiável e válida para a avaliação do design visual de aplicativos móveis criados com uma linguagem de programação baseada em blocos no contexto do ensino de computação na Educação Básica. Foram desenvolvidas também uma rubrica instanciando a abordagem e uma ferramenta automatizando a avaliação segundo a rubrica.

Assim, os resultados satisfazem os objetivos específicos deste trabalho. Conforme os objetivos específicos, o estado da arte foi analisado por meio de um mapeamento sistemático (Objetivo 1). Os resultados do mapeamento indicaram a falta de abordagens para a avaliação de aplicativos móveis criados no contexto educacional que incluam uma avaliação detalhada do design visual e forneçam *feedback* apropriado.

Dados os resultados do mapeamento sistemático, a abordagem foi desenvolvida sistematicamente, definindo os elementos do design visual a serem avaliados com base em diretrizes e princípios do design visual. A abordagem foi instanciada por meio da rubrica CodeMaster UI Design – App Inventor, direcionada à avaliação de aplicativos criados com o App Inventor (Objetivo 2). A avaliação segundo a rubrica foi automatizada, evoluindo a ferramenta CodeMaster 1.0 (Objetivo 3).

A abordagem proposta foi avaliada quanto à confiabilidade e validade por meio de um estudo de caso aplicando a rubrica CodeMaster UI Design – App Inventor (Objetivo 4). De forma geral, os itens apresentaram bons indicadores. Obteve-se um valor satisfatório do alfa de Cronbach ($\alpha = 0,84$) para o conjunto dos 21 itens. Também foram obtidos valores aceitáveis para o carregamento em um fator, com cargas acima de 0,45 para a maioria dos itens. Na análise via TRI, a maioria dos itens apresentou parâmetro de inclinação maior que 1,0 e parâmetros de dificuldade entre -3,0 e 3,5, indicando uma boa calibração.

No decorrer deste trabalho, foram realizadas as seguintes publicações científicas, com os respectivos Qualis – Ciência da Computação:

- SOLECKI, I. S. et al. Automated Assessment of the Visual Design of Android Apps Developed with App Inventor. In: **Proc. of the 51st ACM Technical Symposium on Computer Science Education**. Portland, OR, EUA, 2020. p. 51-57. **(Qualis A1)**
- SOLECKI, I. S. et al. CodeMaster UI design – App Inventor: a rubric for the assessment of the interface design of Android apps developed with app inventor.

In: **Proc. of the 18th Brazilian Symposium on Human Factors in Computing Systems**. Vitória, ES, Brasil, 2019. p. 1-10. **(Qualis B2)**

- SOLECKI, I. S. et al. Estado da Prática do Design Visual de Aplicativos Móveis Desenvolvidos com App Inventor. **Revista Brasileira de Informática na Educação**, v. 28, p. 30-47, 2020. **(Qualis B3)**

Como resultado deste trabalho, está disponível uma abordagem confiável e válida para a avaliação do design visual de aplicativos criados com uma linguagem de programação baseada em blocos, que inclui uma avaliação detalhada do design visual e o fornecimento de *feedback*. A rubrica direcionada ao App Inventor facilita a integração da abordagem em contextos de ensino em que o App Inventor já é utilizado. A abordagem automatizada pode fornecer aos alunos uma avaliação objetiva, além de minimizar o esforço por parte dos professores e auxiliar aqueles que não possuem formação específica em computação. Assim, a abordagem contribui para que o ensino de computação em escolas brasileiras seja mais completo, facilitando a inclusão de competências relacionadas ao design visual de interface de usuário.

Como trabalhos futuros, a avaliação pode ser complementada, aprimorando os critérios existentes ou adicionando critérios cuja automatização envolve identificar a semântica dos componentes ou técnicas de reconhecimento de padrões. Isso inclui critérios para avaliar, por exemplo, se estilos tipográficos adequados foram utilizados conforme a função do texto, ou se os ícones do aplicativo têm um estilo consistente.

Tendo em vista a importância do *feedback* formativo, a apresentação do *feedback* pode ser futuramente aprimorada. A ferramenta atualmente apresenta a rubrica após a avaliação de um aplicativo, permitindo que o aluno identifique, com base nas pontuações obtidas, em quais aspectos o design visual do seu aplicativo pode ser melhorado. Porém, para maximizar o efeito do *feedback*, ele poderia ser apresentado ao aluno de forma personalizada, conforme o seu desempenho, indicando diretamente quais conceitos precisam ser reforçados.

Observando os resultados da avaliação da abordagem, que indicam problemas relacionados principalmente às diretrizes de escrita do Material Design, uma nova avaliação pode ser realizada no contexto da Educação Básica, utilizando aplicativos que representem melhor a aprendizagem desses conceitos. Para essa nova avaliação, os dados podem ser coletados por meio da aplicação de unidades instrucionais que abordem o design visual (por exemplo, oficinas de programação), tendo como público alvo alunos de escolas de Educação Básica. Espera-se que o uso de uma amostra coletada dessa forma resulte em melhoras nos diversos indicadores utilizados nas análises.

Para complementar os resultados da avaliação do design visual, pode ser definida uma nova escala de habilidade, apropriada para o uso no contexto educacional, a partir da estimação dos parâmetros via TRI com dados de contexto educacional. Uma vez que os parâmetros sejam calibrados dessa forma, a atribuição da nota de design visual pode ser feita via TRI e implementada na ferramenta CodeMaster UI Design – App Inventor.

# REFERÊNCIAS


ALLEN, D.; TANNER, K. Rubrics: Tools for making learning goals and evaluation criteria explicit for both teachers and learners. **CBE—Life Sciences Education**, v. 5, n. 3, p. 197-203, 2006.

ALVES, N. C. **CodeMaster:** Um modelo de avaliação do pensamento computacional na educação básica através da análise de código de linguagem de programação visual. 2019. 72 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Catarina, Florianópolis.

ALVES, N. C.; GRESSE VON WANGENHEIM, C.; HAUCK, J. C. R. Approaches to Assess Computational Thinking Competences Based on Code Analysis in K-12 Education: A Systematic Mapping Study. **Informatics in Education**, v. 18, n. 1, p. 17, 2019.

ANDERSON, J. C.; GERBING, D. W. Predicting the performance of measures in a confirmatory factor analysis with a pretest assessment of their substantive validities. **Journal of applied Psychology**, v. 76, n. 5, p. 732, 1991.

ANDRADE, H. G. Understanding rubrics. **Educational leadership**, v. 54, n. 4, p. 14-17, 1997.

ANDRADE, D. F.; TAVARES, H. R.; VALLE, R. C. **Teoria da Resposta ao Item:** Conceitos e Aplicações. São Paulo: Associação Brasileira de Estatística, 2000.

BASILI, V.; CALDIERA, G.; ROMBACH, H. D. Goal question metric paradigm. **Encyclopedia of software engineering**, John Wiley & Sons, p. 528-532, 1994.

BAU, D. et al. Learnable programming: blocks and beyond. **Communications of the ACM**, v. 60, n. 6, 2017. p. 72–80.

BELLER, M. et al. Analyzing the state of static analysis: A large-scale evaluation in open source software. In: **Proc. of the 23rd International Conference on Software Analysis, Evolution, and Reengineering**. Osaka, Japão, 2016. p. 470-481.

BOURQUE, P.; FAIRLEY, R. E. **Guide to the Software Engineering Body of Knowledge, Version 3.0**. IEEE Computer Society, 2014. Disponível em: www.swebok.org. Acesso em: janeiro 2020.

BRANCH, R. M. **Instructional design:** The ADDIE approach. Springer Science & Business Media, 2009.

BROWN, T. A. **Confirmatory factor analysis for applied research**. New York: The Guilford Press, 2006.

CAPELAS, B. Até o fim de 2017, Brasil terá um smartphone por habitante, diz FGV. **O Estadão**, v. 19, n. 04, 2017.

CARMINES, E. G.; ZELLER, R. A. **Reliability and validity assessment**. 5. ed. Beverly Hills: Sage Publications Inc, 1982.

CLARK, L. A.; WATSON, D. Constructing validity: Basic issues in objective scale development. **Psychological assessment**, v. 7, n. 3, p. 309-319, 1995.

CME. **RESOLUÇÃO CME Nº02/2011**. Conselho Municipal de Educação de Florianópolis, 2011.

CNE. **Computação na Escola**. 2019. Disponível em: http://www.computacaonaescola.ufsc.br/. Acesso em: dezembro 2019.

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

COLOR MATTERS. **Basic Color Theory**. 2019a. Disponível em: https://www.colormatters.com/color-and-design/basic-color-theory. Acesso em: dezembro 2019.

COLOR MATTERS. **Color Systems - RGB & CMYK**. 2019b. Disponível em: https://www.colormatters.com/color-and-design/color-systems-rgb-and-cmyk. Acesso em: dezembro 2019.

COMREY, A. L.; LEE, H. B. **A first course in factor analysis**. 2ª ed. Hillsdale NJ: Lawrence: Erlbaum Associates, 1992.

CRONBACH, L. J. Coefficient alpha and the internal structure of tests. **Psychometrika**, v. 16, n. 3. p. 297–334, 1951.

CSTA. **CS Standards**. 2019. Disponível em: https://www.csteachers.org/page/standards. Acesso em: dezembro 2019.

CSTA. **K–12 Computer Science Framework**. 2016. Disponível em: http://www.k12cs.org. Acesso em: dezembro 2019.

DEMETRIO, M. F. **Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação**. Trabalho de Conclusão de Curso. Graduação em Ciência da Computação: Universidade Federal de Santa Catarina, 2017.

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

DEVICEATLAS. **Most used smartphone screen resolutions in 2019**. 2019. Disponível em: https://deviceatlas.com/blog/most-used-smartphone-screen-resolutions. Acesso em: dezembro 2019.

FERREIRA, M. N. F. et al. Ensinando Design de Interface de Usuário na Educação Básica: Um Mapeamento Sistemático do Estado da Arte e Prática. In: **Anais do Workshop de Informática na Escola**. Brasília, DF, Brasil, 2019a. p. 511.

FERREIRA, M. N. F. et al. Learning user interface design and the development of mobile applications in middle school. **ACM Interactions**, v. 26, n. 4, p. 66-69, 2019b.

FERREIRA, M. N. F. et al. Ensinando Design de Interface de Usuário de Aplicativos Móveis no Ensino Fundamental. **Revista Brasileira de Informática na Educação**, v. 28, 2020.

FONTE, D. et al. A flexible dynamic system for automatic grading of programming exercises. In: **Proc. of the 2nd Symposium on Languages, Applications and Technologies**. Porto, Portugal, 2013.

GARNELI, V.; GIANNAKOS, M. N.; CHORIANOPOULOS, K. Computing education in K-12 schools: A review of the literature. In: **Proc. of the 2015 IEEE Global Engineering Education Conference**. Tallinn, Estônia, 2015. p. 543-551.

GARRETT, J. J. **The elements of user experience:** user-centered design for the web and beyond. 2nd ed. New Riders, 2011.

GIL, A. C. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo, 2010.

GLORFELD, L. W. 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, 1995.

GOOGLE. **Core app quality**. 2018a. Disponível em: https://developer.android.com/docs/quality-guidelines/core-app-quality. Acesso em: agosto 2018.

GOOGLE. **Material Design**. 2019. Disponível em: https://material.io/. Acesso em: dezembro 2019.

GOOGLE. **Roboto**. 2018b. Disponível em: https://fonts.google.com/specimen/Roboto. Acesso em: janeiro 2020.

GRESSE VON WANGENHEIM, C. et al. A usability score for mobile phone applications based on heuristics. **International Journal of Mobile Human Computer Interaction**, v. 8, n. 1, p. 23-58, 2016.

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

GROVER, S.; BASU, S.; SCHANK, P. What we can learn about student learning from open-ended programming projects in middle school computer science. In: **Proc. of the 49th ACM Technical Symposium on Computer Science Education**. Baltimore, EUA, 2018. p. 999-1004.

HADDAWAY, N. R. et al. The role of Google Scholar in evidence reviews and its applicability to grey literature searching. **PloS one**, v. 10, n. 9, p. e0138237, 2015.

HOEHLE, H.; ALJAFARI, R.; VENKATESH, V. Leveraging Microsoft′ s mobile usability guidelines: Conceptualizing and developing scales for mobile application usability. **International Journal of Human-Computer Studies**, v. 89, p. 35-53, 2016.

IHANTOLA, P. et al. Review of recent systems for automatic assessment of programming assignments. In: **Proc. of the 10th Koli calling international conference on computing education research**. Koli, Finlândia, 2010. p. 86-93.

INEP. **Censo da Educação Superior 2015**. 2017. Disponível em: http://www.andifes.org.br/wp-content/uploads/2017/04/INEP-Censo-da-Educação-Superior-Andifes-16042017.pdf. Acesso em: março 2020.

INES, G. et al. Evaluation of Mobile Interfaces as an Optimization Problem. **Procedia Computer Science**, v. 112, p. 235-248, 2017.

IRVINE, C.; CROWLEY, K. **Assessment tools in a 21st Century classroom**. 2013. Disponível em: http://etec.ctlt.ubc.ca/510wiki/Assessment_tools_in_a_21st_Century_classroom. Acesso em: dezembro 2019.

JACKSON, J. E. Oblimin Rotation. **Encyclopedia of Biostatistics**, John Wiley & Sons, 2005.

JUSTEN, K. A. **Desenvolvimento de um Analisador de Design de Interface no Contexto do Ensino de Computação com o App Inventor**. Trabalho de Conclusão de Curso. Graduação em Ciência da Computação: Universidade Federal de Santa Catarina, 2019.

LARMAN, C.; BASILI, V. Iterative and incremental developments: a brief history. **IEEE Computer**, v. 36, 2003. p. 47–56.

LEE, D. et al. Antecedents and consequences of mobile phone usability: Linking simplicity and interactivity to satisfaction, trust, and brand loyalty. **Information & Management**, v. 52, n. 3, p. 295-304, 2015.

LIDWELL, W.; HOLDEN, K.; BUTLER, J. **Universal principles of design, revised and updated:** 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design. Rockport Pub, 2010.

LYE, S. Y.; KOH, J. H. L. Review on teaching and learning of computational thinking through programming: What is next for K-12?. **Computers in Human Behavior**, v. 41, p. 51-61, 2014.

MCCAULEY, R. Rubrics as assessment guides. **ACM SIGCSE Bulletin**, v. 35, n. 4, p. 17-18, 2003.

MEC. **Base Nacional Comum Curricular**. 2017. Disponível em: http://basenacionalcomum.mec.gov.br/. Acesso em: janeiro 2020.

MILES, M. B.; HUBERMAN, A. M. **Qualitative data analysis: A sourcebook of new methods**. Sage publications, 1984.

MIT. **MIT App Inventor**. 2019. Disponível em: http://appinventor.mit.edu/. Acesso em: dezembro 2019.

MUSTAFARAJ, E.; TURBAK, F.; SVANBERG, M. Identifying original projects in App Inventor. In: **Proc. of the Thirtieth International Flairs Conference**. Marco Island, FL, Estados Unidos, 2017.

NAGAPPAN, M.; SHIHAB, E. Future trends in software engineering research for mobile apps. In: **Proc. of the 23rd International Conference on Software Analysis, Evolution, and Reengineering**. Osaka, Japão, 2016. p. 21-32.

NIELSEN, J. **Usability engineering**. Elsevier, 1994.

NIELSEN, J. **Weblog Usability:** The Top Ten Design Mistakes. 2005. Disponível em: https://www.nngroup.com/articles/weblog-usability-top-ten-mistakes/. Acesso em: janeiro 2020.

OLSSON, U. Maximum likelihood estimation of the polychoric correlation coefficient. **Psychometrika**, v. 44, n. 4, 1979. p. 443–460.

PASQUALI, L. **Psicometria:** Teoria e aplicações. Brasília, DF: Editora da Universidade de Brasília, 1997.

PATTON, E. W.; TISSENBAUM, M.; HARUNANI, F. MIT App Inventor: Objectives, Design, and Development. In: **Computational Thinking Education**. Springer, Singapore, 2019. p. 31-49.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, v. 64, p. 1-18, 2015.

PORTO, J. V. A.; BARBOSA, H.; GRESSE VON WANGENHEIM, C. Proposta de um Checklist de Avaliação de Usabilidade de Aplicativos Android no Contexto Educacional. In: **Anais do Computer on the Beach**. Florianópolis, Brasil, 2018. p. 870-879.

PPGCC. **Linhas de Pesquisa**. 2019. Disponível em: https://ppgcc.posgrad.ufsc.br/linhas-de-pesquisa-2/. Acesso em: dezembro 2019.

ROGERS, Y.; SHARP, H.; PREECE, J. **Design de Interação**. Bookman, 2013.

SADLER, D. R. Formative assessment and the design of instructional systems. **Instructional science**, v. 18, n. 2, 1989. p. 119- 144.

SALIM, F.; HAQUE, U. Urban computing in the wild: A survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical systems, and Internet of Things. **International Journal of Human-Computer Studies**, v. 81, p. 31-48, 2015.

SALOMÓN, S. et al. A Method for Analyzing the Quality-in-Use in Collaborative Contexts.

In: **Proc. of the XX International Conference on Human Computer Interaction**. Donostia, Espanha, 2019. p. 1-8.

SAMEJIMA, F. A. Estimation of latent ability using a response pattern of graded scores. **Psychometric Monograph**, v. 17, 1969.

SAUNDERS, M.; LEWIS, P.; THORNHILL, A. **Research methods for business students**. 5th ed. Pearson Education India, 2011.

SBC. **Diretrizes para ensino de Computação na Educação Básica**. 2018. Disponível em: https://www.sbc.org.br/documentos-da-sbc/category/131-curriculos-de-referencia. Acesso em: dezembro 2019.

SCHLATTER, T.; LEVINSON, D. **Visual usability: Principles and practices for designing digital applications**. Morgan Kaufmann, 2013.

SESSIONS COLLEGE. **Color Calculator**. 2019. Disponível em: https://www.sessions.edu/color-calculator/. Acesso em: dezembro 2019.

SHERMAN, M.; MARTIN, F. The assessment of mobile computational thinking. **Journal of Computing Sciences in Colleges**, v. 30, n. 6, p. 53-59, 2015.

SOLECKI, I. S. et al. Estado da Prática do Design Visual de Aplicativos Móveis Desenvolvidos com App Inventor. **Revista Brasileira de Informática na Educação**, v. 28, p. 30-47, 2020.

SRIKANT, S.; AGGARWAL, V.. Automatic grading of computer programs: A machine learning approach. In: **Proc. of the 12th International Conference on Machine Learning and Applications**. Miami, EUA, 2013. p. 85-92.

STANKOV, E. et al. A new model for semiautomatic student source code assessment. **Journal of computing and information technology**, v. 21, n. 3, p. 185-194, 2013.

STRAUSS, A.; CORBIN, J. **Basics of qualitative research**. Sage publications, 1990.

TECHNOVATION. **Teacher and Mentor Lesson Guide Lessons 1 - 6**. 2014. Disponível em: https://technovationchallenge.org/wpcontent/uploads/2014/01/TeacherMentorLessonGuide_L1_6.pdf. Acesso em: agosto 2018.

TISSENBAUM, M.; SHELDON, J.; ABELSON, H. From computational thinking to computational action. **Communications of the ACM**, v. 62, n. 3, p. 34-36, 2019.

TROCHIM, W.; DONNELLY, J. P. **Research methods knowledge base**. 3rd ed. Mason, **OH: Atomic Dog**, 2008.

TRUONG, N.; ROE, P.; BANCROFT, P. Static analysis of students' Java programs. In: **Proc. of the 6th Australasian Conference on Computing Education**. Dunedin, Nova Zelândia, 2004. p. 317-325.

W3C. **Understanding Success Criterion 1.3.4: Orientation**. 2017. Disponível em: https://www.w3.org/WAI/WCAG21/Understanding/orientation.html. Acesso em: dezembro 2019.

W3C. **Web Content Accessibility Guidelines 2.1**. 2018. Disponível em: https://www.w3.org/TR/2018/REC-WCAG21-20180605/. Acesso em: dezembro 2019.

WAGNER, A. et al. Using app inventor in a K-12 summer camp. In: **Proc. of the 44th ACM technical symposium on Computer science education**. Denver, EUA, 2013. p. 621-626.

WANG, M.; LI, X. Effects of the aesthetic design of icons on app downloads: evidence from an android market. **Electronic Commerce Research**, v. 17, n. 1, p. 83-102, 2017.

WASSERMAN, A. Software engineering issues for mobile application development. In: **Proc. of the FSE/SDP workshop on Future of software engineering research**. Nova York, NY, Estados Unidos, 2010. p. 397-400.

WEBAIM. **Contrast Checker**. 2019. Disponível em: https://webaim.org/resources/contrastchecker/. Acesso em: dezembro 2019.

WHITTAKER, C. R.; SALEND, S. J.; DUHANEY, D. Creating instructional rubrics for inclusive classrooms. **Teaching Exceptional Children**, v. 34, n. 2, p. 8-13, 2001.

WIKIPEDIA. **A smartphone positioned upright (portrait orientation) and horizontally (landscape orientation)**. 2012. Disponível em: https://en.wikipedia.org/wiki/Page_orientation#/media/File:800x518_smartphone_portrait_vs_landscape_orientation.png. Acesso em: dezembro 2019.

WILIAM, D. What is assessment for learning?. **Studies in educational evaluation**, v. 37, n. 1, p. 3-14, 2011.

WOHLIN, C. et al. **Experimentation in software engineering**. Springer Science & Business Media, 2012.

WOLBER, D. App Inventor and Real-World Motivation. In: **Proc. of the 42nd ACM technical symposium on Computer science education**. Dallas, TX, Estados Unidos, 2011. p. 601-606.

WOLBER, D.; ABELSON, H.; FRIEDMAN, M. Democratizing computing with app inventor. **GetMobile: Mobile Computing and Communications**, v. 18, n. 4, p. 53-58, 2015.

XU, C.; PEAK, D.; PRYBUTOK, V. A customer value, satisfaction, and loyalty perspective of mobile application recommendations. **Decision Support Systems**, v. 79, p. 171-183, 2015.

YIN, R. K. **Estudo de Caso:** Planejamento e métodos. 5. ed. Bookman, 2015.

ZAPATA, B. C. et al. Mobile PHRs compliance with Android and iOS usability guidelines. **Journal of medical systems**, v. 38, n. 8, p. 81, 2014.

# APÊNDICE A – Propriedades relacionadas à aparência dos componentes do App Inventor

A Tabela A-1 indica quais propriedades relacionadas à aparência (relevantes para a avaliação do design visual) cada componente visual do App Inventor possui. O valor padrão das propriedades (valor que a propriedade possui quando o componente é criado) é indicado pois é necessário conhecê-lo para realizar a análise dos aplicativos. Isso porque, no caso de várias propriedades, quando o valor padrão é usado, nenhum valor aparece no código fonte (arquivo .scm).

Tabela A-1 – Propriedades dos componentes do App Inventor.

| **Propriedade** | **Componentes** | **Valor padrão** |
|---|---|---|
| AccentColor | Screen1* | #ff4081ff** |
| Altura | Botão, CaixaDeSeleção, EscolheData, Imagem, Legenda, EscolheLista, VisualizadorDeListas, CaixaDeSenha, Switch, CaixaDeTexto, EscolheHora, NavegadorWeb, OrganizaçãoHorizontal, HorizontalScrollArrangement, OrganizaçãoEmTabela, OrganizaçãoVertical, VerticalScrollArrangement, EscolheImagem, ReprodutorDeVídeo, Pintura, SpriteImagem, Map, EscolheContato, EscolheEmail, EscolheNúmeroDeTelefone | Automático |
| CadeiaDeElementos | EscolheLista, VisualizadorDeListas, ListaSuspensa | *Vazio* |
| CorÀDireita | Deslizador | #ffc800ff |
| CorÀEsquerda | Deslizador | #888888ff |
| CorDeFundo | Tela | #ffffffff |
| | CaixaDeSeleção, Legenda, Switch | #ffffff00/Nenhum |
| | Botão, EscolheData, EscolheLista, EscolheHora, EscolheImagem, EscolheContato, EscolheNúmeroDeTelefone | *Degradê, aproximadamente #d8d8d8ff* |
| | VisualizadorDeListas, CaixaDeSenha, CaixaDeTexto, OrganizaçãoHorizontal, HorizontalScrollArrangement, OrganizaçãoVertical, VerticalScrollArrangement, Pintura, EscolheEmail | #000000ff |
| | Notificador | #444444ff |
| CorDeFundoDoItem | EscolheLista | #ffffffff |
| CorDePintura | Bola, Pintura | #000000ff |
| CorDeSeleção | VisualizadorDeListas | #ccccccff |
| CorDeTexto | Botão, Switch, CaixaDeSeleção, EscolheData, Legenda, EscolheLista, CaixaDeSenha, CaixaDeTexto, EscolheHora, EscolheImagem, EscolheContato, EscolheEmail, EscolheNúmeroDeTelefone | #000000ff |
| | VisualizadorDeListas, Notificador | #ffffffff |
| CorDeTextoDoItem | EscolheLista | #ffffffff |
| Dica | CaixaDeSenha | *Vazio* |
| | CaixaDeTexto, EscolheEmail | *Texto padrão (ex.: "Dica para CaixaDeTexto1")* |

| | | |
|---|---|---|
| FamíliaDaFonte | Botão, CaixaDeSeleção, EscolheData, Legenda, EscolheLista, CaixaDeSenha, Switch, CaixaDeTexto, EscolheHora, EscolheImagem, EscolheContato, EscolheEmail, EscolheNúmeroDeTelefone | Padrão (geralmente, equivale a sem serifa) |
| FonteItálico | Botão, CaixaDeSeleção, EscolheData, Legenda, EscolheLista, CaixaDeSenha, Switch, CaixaDeTexto, EscolheHora, EscolheImagem, EscolheContato, EscolheEmail, EscolheNúmeroDeTelefone | Desativado |
| FonteNegrito | Botão, CaixaDeSeleção, EscolheData, Legenda, EscolheLista, CaixaDeSenha, Switch, CaixaDeTexto, EscolheHora, EscolheImagem, EscolheContato, EscolheEmail, EscolheNúmeroDeTelefone | Desativado |
| Forma | Botão, EscolheData, EscolheLista, EscolheHora, EscolheImagem, EscolheContato, EscolheNúmeroDeTelefone | Padrão |
| Imagem | Botão, EscolheData, EscolheLista, EscolheHora, EscolheImagem, EscolheContato, EscolheNúmeroDeTelefone, OrganizaçãoHorizontal, HorizontalScrollArrangement, OrganizaçãoVertical, VerticalScrollArrangement, SpriteImagem, Marker, Imagem | Nenhuma |
| ImagemDeFundo | Pintura, Tela | Nenhuma |
| Largura | Botão, CaixaDeSeleção, EscolheData, Imagem, Legenda, EscolheLista, VisualizadorDeListas, CaixaDeSenha, Deslizador, ListaSuspensa, Switch, CaixaDeTexto, EscolheHora, NavegadorWeb, OrganizaçãoHorizontal, HorizontalScrollArrangement, OrganizaçãoEmTabela, OrganizaçãoVertical, VerticalScrollArrangement, EscolheImagem, ReprodutorDeVídeo, Pintura, SpriteImagem, Map, EscolheContato, EscolheEmail, EscolheNúmeroDeTelefone | Automático |
| Mensagem | ListaSuspensa | *Vazio* |
| PrimaryColor | Screen1 | #3f51b5ff |
| PrimaryColorDark | Screen1 | #303f9fff |
| Raio | Bola | 5 |
| TamanhoDaFonte | Botão, CaixaDeSeleção, EscolheData, Legenda, EscolheLista, CaixaDeSenha, Switch, CaixaDeTexto, EscolheHora, EscolheImagem, Pintura, EscolheContato, EscolheEmail, EscolheNúmeroDeTelefone | 14 |
| TamanhoDoTexto | VisualizadorDeListas | 22 |
| Texto | Botão, Switch, CaixaDeSeleção, EscolheData, Legenda, EscolheLista, EscolheHora, EscolheImagem, EscolheContato, EscolheNúmeroDeTelefone | *Texto padrão (ex.: "Texto para Botão1")* |
| | CaixaDeSenha, CaixaDeTexto, EscolheEmail | *Vazio* |
| ThumbColorActive | Switch | #ffffffff |
| ThumbColorInactive | Switch | #ccccccff |
| Título | Tela | *Nome atribuído à tela* |
| | EscolheLista | *Vazio* |
| TelaSobre | Tela | *Vazio* |
| TrackColorActive | Switch | #00ff00ff |
| TrackColorInactive | Switch | #444444ff |

* *Screen1* indica propriedades que existem apenas na primeira tela (sempre chamada de Screen1)

** Representação hexadecimal da cor, conforme apresentada no App Inventor

Fonte: Elaborada pelo autor

# APÊNDICE B – Implementação dos critérios

A seguir, é apresentada uma descrição do funcionamento dos critérios mais elaborados da rubrica. A implementação da maioria dos critérios apenas detecta o valor das propriedades dos componentes e verifica se satisfazem as condições definidas pela rubrica. Assim, apenas os algoritmos dos critérios mais elaborados são apresentados e descritos por meio de pseudocódigos a seguir. Os algoritmos têm como entrada uma lista de objetos JSON, na qual cada objeto representa uma tela do aplicativo, e têm como saída a pontuação do aplicativo no critério.

Na definição e descrição dos critérios, o termo "botões" se refere não apenas ao componente Botão, mas também aos componentes EscolheData, EscolheLista, EscolheHora, EscolheImagem, EscolheContato e EscolheNúmeroDeTelefone, que também são botões, porém com um propósito especial.

## Critério SistemaDeCores

A função avaliarSistemaDeCores (Figura B-1) inicialmente detecta as cores definidas por meio de propriedades (p. ex. cor de fundo de botões) (linha 1) e, em seguida, se houver imagens no aplicativo, detecta as cores predominantes de cada imagem (linha 2). Então, a pontuação é determinada conforme definido na rubrica (linhas 3 a 9).

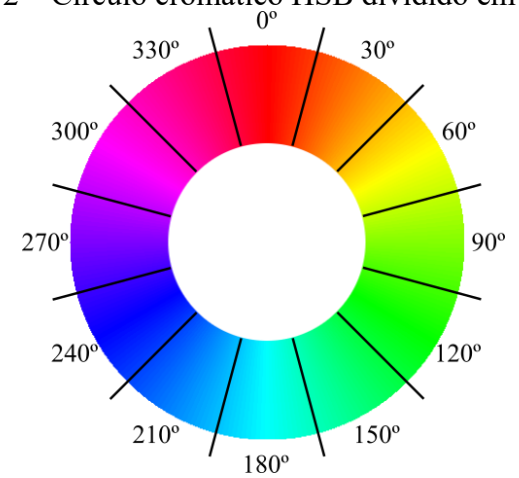
Figura B-1 – Algoritmo do critério SistemaDeCores.

```
   FUNÇÃO avaliarSistemaDeCores
   Entrada: Lista T de objetos JSON representando as telas do aplicativo
   Saída: Pontuação do aplicativo no critério
   // Detecta as cores definidas por meio de propriedades
1  coresProp ← detectarCoresDePropriedades(T)
   // Detecta as cores predominantes das imagens usadas no aplicativo
2  coresImg ← detectarCoresDeImagens(T)
   // Opa
3  coresUsadas ← União de coresProp e coresImg
4  Se coresUsadas.tamanho = 1 ou coresUsadas.tamanho = 2
5          Retorna 2
6  Senão, se coresUsadas.tamanho = 3
7          Retorna 1
8  Senão // 4 ou mais cores, ou nenhuma cor
9          Retorna 0
```

Fonte: Elaborada pelo autor.

Para detectar quantas cores o aplicativo possui, utiliza-se o sistema de cores HSB (*hue*, *saturation*, *brightness*). O sistema HSB[1] é uma representação do sistema RBG que possibilita dispor as cores em um círculo cromático de acordo com o matiz (*hue*), que varia de 0º a 360º. O círculo cromático é dividido em 12 faixas, e cada uma das faixas é considerada uma cor (Figura B-2). Dessa forma, o aplicativo pode ter de 0 a 12 cores. Esse número foi escolhido pois o círculo cromático tipicamente é dividido nessa quantidade de cores (SCHLATTER; LEVINSON, 2013).

Figura B-2 – Círculo cromático HSB dividido em 12 faixas.




Cada faixa é identificada pelo matiz (*hue* do sistema HSB) central da faixa. As cores básicas do sistema RGB – vermelho, verde e azul – possuem, respectivamente, os matizes 0º, 120º e 240º.
Fonte: Elaborada pelo autor.

A função detectarCoresDePropriedades não é apresentada em pseudocódigo pois seu funcionamento é mais facilmente apresentado como descrito a seguir. De forma geral, a função apenas extrai das estruturas JSON as propriedades que definem cores (ver Apêndice A) e identifica a qual das 12 faixas cada cor extraída pertence conforme seu matiz (ver Figura B-2). No entanto, há alguns casos especiais na detecção:

- Preto, branco e tons de cinza (ou seja, cores com baixa saturação) são ignorados.
- A cor de fundo de botões e de organizadores, quando esses possuem uma imagem, não aparece no aplicativo. Portanto, essas cores são ignoradas.
- Se a cor não está definida no JSON, a cor padrão é usada (ver Apêndice A). Se a cor padrão não for preto/branco/cinza, ela é considerada.

[1] https://en.wikipedia.org/wiki/HSL_and_HSV

A função detectarCoresDeImagens (Figura B-3) inicialmente lista todas as imagens usadas (linha 1) e, antes de analisar uma imagem, a reduz se for muito grande, diminuindo o processamento necessário (linha 4).

Figura B-3 – Pseudocódigo da função detectarCoresDeImagens.

```
FUNÇÃO detectarCoresDeImagens
Entrada: Lista T de objetos JSON representando as telas do aplicativo
Saída: Conjunto de índices (de 1 a 12) das cores usadas
1  Imagens ← Lista todas as imagens usadas em T
2  coresDasImagens ← Conjunto vazio
3  Para cada img em Imagens
4      Redimensiona img se for muito grande (largura > 480 ou altura > 480)
       // Representa a divisão do círculo cromático em 36 (12*3) subfaixas
5      subfaixas ← Arranjo de tamanho 36
       // Croma é o C do sistema de cores HCL
6      pixelsColoridos ← Conjunto de pixels coloridos (croma > 20.0) de img
7      Para cada i de 1 até 36
8          subfaixas[i] ← Quantos pixels de pixelsColoridos pertencem à subfaixa i
9      grupos ← Lista todos os conjuntos possíveis de 3 elementos adjacentes de subfaixas
       // A quantidade de pixels de um grupo é quantos pixels as 3 subfaixas dele têm ao todo
10     Ordena grupos em ordem decrescente de quantidade de pixels
       // Os grupos escolhidos representam as cores predominantes da imagem
11     gruposEscolhidos ← Conjunto vazio
       // Devido à ordenação, trata grupos com mais pixels primeiro
12     Para cada g em grupos
           // Porcentagem que o grupo tem de todos os pixels coloridos da imagem
13         pctg ← (Quantidade de pixels de g) / pixelsColoridos.tamanho
           // Só detecta cores com
14         Se pctg > PORCENTAGEM_MÍNIMA e as subfaixas de g não pertencem a nenhum grupo já
             adicionado a gruposEscolhidos
15             gruposEscolhidos.adicionar(g)
           // Define uma das 3 subfaixas como "principal" (usa uma média ponderada entre o matiz das
             subfaixas; os pesos são a porcentagem de pixels de cada subfaixa no grupo)
16         subfaixaPrincipal ← subfaixaPrincipal(g)
17         cor ← Cor correspondente à faixa em que subfaixaPrincipal está
           // Adiciona a cor que esse grupo representa
18         coresDasImagens.adicionar(cor)
19 Retorna coresDasImagens
```

Fonte: Elaborada pelo autor.

Para detectar as cores em uma imagem, o círculo cromático também é dividido em 12 faixas, como indicado na Figura B-2. Porém, observa-se que, em uma imagem, uma tonalidade predominante pode apresentar variações e "invadir" outra tonalidade, não seguindo a divisão do círculo cromático em faixas. Portanto, a função identifica essa situação, na qual uma cor predominante está simultaneamente em duas faixas, e detecta apenas uma cor em vez de duas.

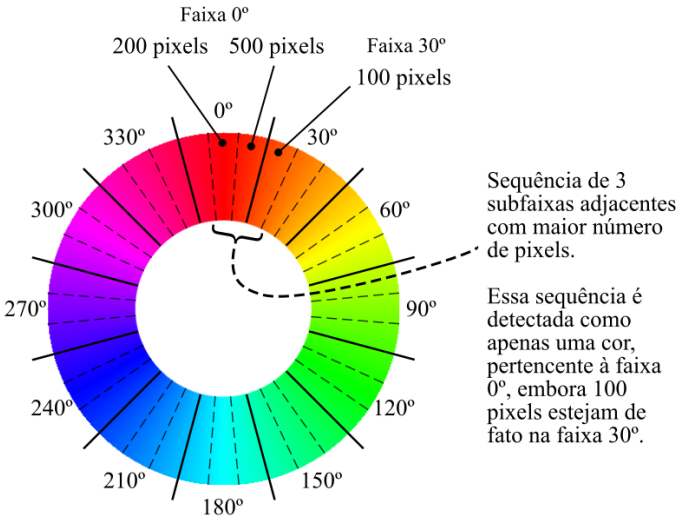
Para detectar essas pequenas variações de tonalidade, cada faixa é subdividida em 3 subfaixas (linha 5). Então, a função varre a imagem, detectando a cor de cada pixel colorido e determinando à qual subfaixa ele pertence (linhas 6 a 8).

Para determinar quais pixels são considerados coloridos, as cores são convertidas para o sistema HCL (*hue*, *chroma*, *luminance*), no qual as cores têm uma distribuição mais uniforme na percepção humana[2]. A coloração é determinada pelo componente croma, sendo que 0,0 representa saturação mínima (cinza), e valores maiores representam coloração mais intensa. Assim, com base na observação de diferentes valores de croma, são considerados coloridos apenas os pixels com croma acima de 20,0.

Por fim, as subfaixas são agrupadas de 3 em 3 (linha 9), e os grupos com maior porcentagem de pixels determinam as cores predominantes da imagem (linhas 10 a 19). Apenas grupos com porcentagem suficiente (maior do que *PORCENTAGEM_MÍNIMA*) são considerados (linhas 14 e 15). Assim, cores que aparecem de forma mais sutil na imagem não são detectadas.

O valor de *PORCENTAGEM_MÍNIMA* deve ser menor do que 1/12 (8,3%) para que seja possível detectar corretamente as cores em uma imagem com todas as cores igualmente distribuídas. Em uma imagem assim, se a porcentagem mínima fosse maior do que 1/12, nenhuma cor seria detectada, pois todas teriam porcentagem insuficiente. Portanto, o valor escolhido foi *PORCENTAGEM_MÍNIMA* = 8%.

Figura B-4 – Exemplo de funcionamento da função detectarCoresDeImagens.




Fonte: Elaborada pelo autor.

[2] http://hclwizard.org/

Um exemplo do funcionamento do algoritmo é apresentado na Figura B-4. O exemplo considera uma imagem que possui 800 pixels coloridos, distribuídos entre a primeira e a segunda faixa. Devido à proximidade dos matizes, apenas uma cor é detectada, pertencente à primeira faixa, em vez de duas cores.

### Critério CoresHarmônicas

A função avaliarCoresHarmônicas (Figura B-5), de forma similar ao critério SistemaDeCores, inicialmente detecta todas as cores definidas por meio de propriedades (linha 1). Essa detecção é igual à detecção feita para o critério SistemaDeCores, exceto que as cores em si são listadas em vez de serem categorizadas em faixas do círculo cromático. Este critério não avalia cores de imagens devido à dificuldade de harmonizar cores de imagens.

Figura B-5 – Algoritmo do critério CoresHarmônicas

```
   FUNÇÃO avaliarCoresHarmônicas
   Entrada: Lista T de objetos JSON representando as telas do aplicativo
   Saída: Pontuação do aplicativo no critério
1  cores ← detectarCoresDosComponentes(T)
2  Para cada cor1 em cores
3      Para cada cor2 diferente de cor1 em cores
4          distância ← Distância (em graus) entre os matizes de cor1 e cor2
5          sãoHarmônicas ← Falso
6          Se cor1 e cor2 são variantes segundo a paleta de cores do Material Design
7              sãoHarmônicas ← Verdadeiro
           // Variante, análoga, complementar ou triádica
8          Senão, se distância < 35 ou pertence a um dos intervalos [170, 180], [110, 130]
9              sãoHarmônicas ← Verdadeiro
           // Análoga da análoga
10         Senão, se distância < 70
               // Só considera como análoga da análoga se houver uma cor entre as duas
11             central ← Matiz exatamente no meio dos matizes de cor1 e cor2
12             Se existe alguma cor em cores com matiz no intervalo [central – 5, central + 5]
13                 sãoHarmônicas ← Verdadeiro
           // Encontrou um par não harmônico – não pontua
14         Se sãoHarmônicas = Falso
15             Retorna 0
   // Nenhum par não harmônico foi encontrado – pontua
16 Retorna 2
```

Fonte: Elaborada pelo autor.

Após detectar todas as cores usadas, o algoritmo verifica se todas elas são harmônicas entre si, ou seja, cada cor usada deve manter uma relação harmônica com todas as outras. As relações de harmonia são definidas pela distância dos matizes. Portanto, para todos os pares

possíveis entre as cores detectadas, a distância entre os seus matizes é calculada (linhas 2 a 4). Uma vez que as cores estão em um círculo, há duas distâncias possíveis (no sentido horário e no anti-horário), e a menor delas é considerada.

Para que o par de cores seja considerado harmônico, a distância deve satisfazer uma das seguintes condições:

- *distância* ≤ 35º (variante ou análoga);
- 170 ≤ *distância* ≤ 180 (complementar);
- 110 ≤ *distância* ≤ 130 (triádica);
- 35º < *distância* ≤ 70º (análoga da análoga).

Esses limites foram definidos com base no valor que define a relação de harmonia e uma tolerância (Tabela B-1). Considerando a divisão típica do círculo cromático em 12 cores, seriam consideradas variantes cores com diferença de matiz de 30º. Porém, visto que não se encontrou uma definição rígida de qual distância define uma cor análoga, distâncias menores também são aceitas. Assim, para qualquer valor menor do que 35º (30º mais tolerância de 5º), a cor é considerada como variante ou análoga. No caso da análoga da análoga, deve existir uma cor aproximadamente no centro das duas (com tolerância de 5º, análoga de ambas) para que realmente se considere análoga da análoga.

Tabela B-1 – Valores que definem as relações de harmonia e tolerâncias utilizadas.

| **Relação** | **Distância de matizes** | **Tolerância** | **Intervalo resultante*** |
|---|---|---|---|
| Variante | 0º | 10º | [0, 10] |
| Análoga | 30º | 5º | [25, 35] |
| Complementar | 180º | 10º | [170, 180] |
| Triádica | 120º | 10º | [110, 130] |

* Os limites são *distância* ± *tolerância*, limitados ao intervalo [0, 180]

Fonte: Elaborada pelo autor.

Uma exceção a essas regras são cores definidas na paleta de cores do Material Design (linhas 6 e 7). Em alguns casos, a paleta do Material Design define como variantes cores com grande diferença de matiz, não respeitando esses limites, mas esses casos também são detectados como variantes pelo algoritmo.

Se houver algum par de cores que não satisfaz nenhuma dessas condições, o aplicativo recebe pontuação 0 (linhas 5 a 15). Caso contrário, recebe pontuação 2 (linha 16).

### Critério ÍconesMD

A função avaliarÍcones (Figura B-6) inicialmente lista todos os componentes que são considerados ícones, ou seja, botões com imagem e sem texto (linha 1). Se não houver ícones (ou seja, se nenhum componente assim for encontrado), o aplicativo recebe NA (linhas 2 e 3). Caso contrário, o algoritmo compara a imagem de cada um desses componentes com os ícones do Material Design utilizando a biblioteca OpenCV 3.4.3[3] (linhas 5 a 15). O Material Design oferece cada ícone em 5 temas diferentes, nas cores preto e branco. Portanto, o conjunto de ícones usado (linha 4) inclui ícones nos 5 temas e em ambas as cores.

Figura B-6 – Algoritmo do critério ÍconesMD.

```
   FUNÇÃO avaliarÍcones
   Entrada: Lista T de objetos JSON representando as telas do aplicativo
   Saída: Pontuação do aplicativo no critério
1  botõesÍcones ← Lista todos os botões com imagem e sem texto em T
   // O critério não se aplica
2  Se botõesÍcones.tamanho = 0
3      Retorna NÃO_SE_APLICA
4  íconesMD ← Conjunto de ícones do Material Design
5  Para cada botão em botõesÍcones
6      img ← botão.imagem convertida para escala de cinza
7      Se img possui transparência
8          // Cria duas versões, com fundo preto e com fundo branco
9          comFundoBranco ← img sobre fundo branco
10         comFundoPreto ← img sobre fundo preto
11         Se não existe ícone em íconesMD que satisfaz (éSimilar(ícone, comFundoBranco) ou
             éSimilar(ícone, comFundoPreto))
12             Retorna 0
13     Senão // Não possui transparência
14         Se não existe ícone em íconesMD que satisfaz éSimilar(ícone, img)
15             Retorna 0
   // Todos as imagens em botões são similares a algum ícone do Material Design
16 Retorna 2
```

Fonte: Elaborada pelo autor.

A função do OpenCV utilizada para a comparação de imagens apresentou resultados melhores convertendo as imagens para escala de cinza (linha 6). Além disso, os resultados são melhores para imagens sem transparência. Portanto, os ícones do Material Design foram baixados no formato PNG, e a transparência foi trocada por fundo branco nos ícones pretos e por fundo preto nos ícones brancos. De forma similar, se uma imagem possui transparência, o

[3] https://opencv.org/

algoritmo cria duas versões sem transparência, uma com fundo preto e outra com fundo branco, e apenas uma das duas precisa ser similar a algum ícone do Material Design (linhas 7 a 12). Se a imagem não possui transparência, não é necessário fazer isso (linhas 13 a 15). Se alguma das imagens verificadas não for similar a nenhum ícone, ela não é um ícone do Material Design, e a pontuação é 0, conforme definido na rubrica; caso contrário, todas as imagens são ícones do Material Design, e a pontuação é 2.

A comparação das imagens com os ícones (linhas 11 e 14) é feita pela função éSimilar (Figura B-7). Essa função tem como entrada duas imagens: um ícone e uma imagem para ser comparada com esse ícone. Inicialmente, a segunda imagem é redimensionada para o tamanho do ícone (linha 1). Em seguida, a função matchTemplate do OpenCV é utilizada para calcular a similaridade da imagem com o ícone.

Figura B-7 – Pseudocódigo da função éSimilar.

```
FUNÇÃO éSimilar
Entrada: Duas imagens, ícone e img
Saída: Se as imagens são similares
1 redimensionada ← img redimensionada para o tamanho de ícone
  // Usa a função matchTemplate da biblioteca OpenCV 3.4.3
2 ccoeff ← Nível de similaridade usando o algoritmo CCOEFF normalizado
3 ccorr ← Nível de similaridade usando o algoritmo CCORR normalizado
4 sqdiff ← Nível de similaridade usando o algoritmo SQDIFF normalizado
  // Valores escolhidos com base em testes
5 Se ccoeff ≥ 0,8 e ccorr ≥ 0,9 e sqdiff ≤ 0,05
6       Retorna Verdadeiro
7 Senão
8       Retorna Falso
```

Fonte: Elaborada pelo autor.

O OpenCV possui 6 opções de algoritmos para a função matchTemplate (CCOEFF, CCORR, SQDIFF e uma versão denominada “normalizada” de cada um desses). Os melhores resultados foram obtidos usando os 3 algoritmos normalizados juntos (linhas 2 a 4). Os valores de limite (linha 5) foram definidos por meio de testes, de forma que, em geral, as imagens são consideradas similares se:

- A imagem é exatamente a mesma imagem que o ícone;
- A imagem é o ícone com qualidade menor (p. ex. salvo no formato JPG em vez de PNG);
- A imagem é um ícone de outro conjunto, mas muito parecido com um ícone do Material Design;
- A imagem é um dos ícones do Material Design, mas com cores diferentes.

O conjunto de ícones usado para a avaliação (Figura B-6, linha 4) idealmente incluiria todos os ícones em todos os temas[4]. No entanto, muitos ícones são muito similares (ou até iguais) entre os temas. Portanto, é utilizado um conjunto de ícones minimizado para reduzir o processamento ao avaliar o critério. Quando os ícones foram baixados, havia 1.044 ícones, totalizando 10.440 ícones nos 5 temas e 2 cores, e foi possível reduzir esse conjunto a 2.794 ícones.

Para montar esse conjunto minimizado, foi utilizado o algoritmo apresentado na Figura B-8. Esse algoritmo também verifica a similaridade dos ícones usando a função éSimilar, já apresentada na Figura B-7.

Figura B-8 – Algoritmo para montar conjunto minimizado de ícones.

```
FUNÇÃO gerarConjuntoMinimizado
Entrada: Conjunto Ícones de ícones do Material Design
Saída: Conjunto minimizado
1 minimizado ← Conjunto vazio
2 Para cada ícone em Ícones
      // Só adiciona se ainda não há nenhum ícone similar a este no conjunto
3     Se não existe outroÍcone em minimizado que satisfaz éSimilar(ícone, outroÍcone)
4         minimizado.adicionar(ícone)
5 Retorna minimizado
```

Fonte: Elaborada pelo autor.

Ressalta-se que o critério deveria ter sido definido de outra forma, pois deveria verificar se *todos os ícones* usados no aplicativo são ícones do Material Design. Ícones podem ser usados não apenas em botões, mas também como elementos não clicáveis, de forma similar a legendas. Além disso, nem todas imagens usadas em botões são ícones. Assim, deveriam ser verificados todos os componentes com imagem do aplicativo, identificando quais imagens são ícones e quais não são (p. ex. fotos), de forma a aplicar o critério apenas a ícones. Não foi possível implementar essa distinção, mas o critério foi mantido na rubrica considerando a possibilidade de alterá-lo futuramente.

[4] Disponíveis em https://material.io/resources/icons/