Universidade Federal de Santa Catarina

Evolução de um Modelo de Avaliação de Design de Interface no Contexto do Ensino de Computação com o App Inventor

Gabriel Andrade Borges Nascimento

Martina Klippel Brehm

Florianópolis

2022.2

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Evolução de um Modelo de Avaliação de Design de Interface no Contexto do Ensino de Computação com o App Inventor

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do Grau de Bacharelado em Sistemas de Informação.

Alunos: Gabriel Andrade Borges Nascimento
Martina Klippel Brehm

Orientadora: Christiane Gresse von Wangenheim
Co-orientador Jean C. R. Hauck

Florianópolis
2022.2

Agradecimentos

Gostaria de expressar minha profunda gratidão a todos que contribuíram para a realização deste trabalho.

Agradeço imensamente a orientadora, professora Christiane, por sua orientação, paciência e contribuições durante todas as etapas deste trabalho.

Ao GQS/UFSC, ao professor Jean e ao Adriano pela disponibilidade, pelas sugestões e suas contribuições que enriqueceram este projeto.

À UFSC, por fornecer os recursos necessários e um ambiente propício para a realização deste estudo.

À minha família, pelo apoio incondicional e pelo incentivo constante.

A todos, meu sincero agradecimento.

Resumo

A tecnologia da informação teve uma grande expansão em todo mundo nas últimas décadas, tornando-se consequentemente algo de grande relevância no dia a dia das pessoas. Por causa disso, para acompanhar o desenvolvimento mundial, a inclusão de princípios básicos da computação na educação básica torna-se importante para incentivar o interesse dos jovens pela carreira de computação e também para adquirirem conhecimentos como usuários de tecnologia. Uma maneira de ensinar computação é pelo desenvolvimento de aplicativos para dispositivos Android utilizando App Inventor, um ambiente de programação baseado em blocos. Frequentemente o foco do ensino de computação com o App Inventor está nos conceitos típicos de programação. No entanto, o desenvolvimento de aplicativos envolve conceitos além da codificação, como o design de interface de usuário (UI). Considerando a importância da usabilidade e do design de interface para apps, torna-se importante ensinar não apenas programação, mas também conceitos básicos de design de interface. *Feedback* por meio de avaliações auxiliam no aprendizado dos estudantes, porém podem se tornar uma tarefa complexa e trabalhosa para os professores. Este trabalho pode ser automatizado pelo CodeMaster-UI design que analisa automaticamente os projetos de apps dos alunos e o design de interface fornecendo *feedback* utilizando uma rubrica. Os critérios de avaliação na rubrica foram criados com base em guias de estilo, porém como esses guias evoluíram, o objetivo deste trabalho é analisar o estado da arte, evoluir e melhorar o modelo de avaliação de CodeMaster UI Design, implementando os ajustes necessários e avaliando sua confiabilidade e validade. Os resultados da análise do coeficiente de correlação intraclasse (ICC3) indicaram um valor de 0,6791, o que sugere uma correlação boa entre as avaliações dos aplicativos. Além disso, a análise de Bland-Altman revelou uma diferença média de 1,102 unidades entre as avaliações dos dois avaliadores, com limites de concordância entre -2,3849 (inferior) e 4,5889 (superior). Esses resultados indicam que ainda há espaço para melhorias no processo de avaliação. Como resultados obtidos, foi possível implementar as melhorias no modelo de avaliação do CodeMaster-UI Design, apresentando bons resultados nos testes de corretude. Com isso, espera-se que essa evolução contribua significativamente para a ampliação do ensino de algoritmos, programação e design de interface na educação

básica, facilitando ainda mais o ensino e a popularização desses conhecimentos no Brasil.

Sumário

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Sistemas de software atualmente são parte fundamental de muitas atividades do dia-a-dia (Salim *et al*., 2015) e o número de pessoas que estão utilizando as aplicações móveis tem aumentado consideravelmente (Valentim *et al*., 2015). Deste modo, é fundamental que as pessoas possam ter conhecimentos básicos de computação desde o início da vida escolar (Oliveira *et al*., 2014).

Isso também é indicado no Brasil. A Base Nacional Comum Curricular (BNCC) indica que os jovens devem ser preparados com novos conhecimentos inerentes a uma sociedade e profissões em constante mudança. Entre esses conhecimentos está a computação e tecnologias digitais, que inclui habilidades relacionadas à computação para o Ensino Médio (Ministério da Educação, 2022).

O crescimento e o desenvolvimento de iniciativas que buscam incorporar as tecnologias da informação e comunicação ao cenário da educação brasileira vive um processo constante de atualização (Tunin *et al*., 2022). Uma dessas iniciativas é a Computação na Escola/INCoD/INE/UFSC[1] que visa o ensino de design de interface junto com a programação na educação básica (Ferreira *et al*., 2019).

Neste estágio escolar, é muito comum a utilização de linguagens de programação baseadas em blocos, pois facilitam o foco na estrutura e na lógica do projeto quando comparadas com linguagens baseadas em texto, que exigem conhecimento de sintaxe (Kelleher *et al*., 2005). Um exemplo de um ambiente de programação baseado em blocos é o App Inventor[2]. A criação de aplicativos com o App Inventor é intuitiva e exige menos conhecimento prévio em programação. O App Inventor propicia um ambiente de aprendizagem baseado no construcionismo, uma vez que permite aos estudantes criarem aplicações à medida que descobrem e exercitam sua criatividade, tornando o aprendizado mais lúdico (Patton *et al*., 2019). Adotando o ciclo de *use-modify-create (*Lee *et al*., 2011*)*, principalmente no nível de *create*, os alunos são estimulados a criar os seus próprios aplicativos adotando uma estratégia de ensino baseado em problemas.

[1] www.computacaonaescola.ufsc.br
[2] www.appinventor.mit.edu

Assim, considerando que a usabilidade e o design de interface estão entre os importantes fatores de qualidade de aplicativos móveis, o ensino de computação por meio do desenvolvimento de apps com App Inventor, permite também ensinar conceitos de design de interface (Ferreira *et al.*, 2019). O ensino de conceitos de design de interface pode ser alinhado às diretrizes de guias de estilo como o proposto pelo *Material Design* (2022), voltado ao design de interfaces para aplicativos Android fornecendo recomendações para o design de diversos elementos de interface, como cores, fontes, layout e componentes.

Porém existe uma lacuna de suporte para a avaliação da aprendizagem e o seu *feedback* para o estudante (Falkembach *et al*., 2003). A avaliação e *feedback* instrucional são muito importantes para que o processo de ensino e aprendizagem seja bem sucedido (Hattie *et al.*, 2007). No contexto de aprendizagem baseada em problemas, em que não existe um gabarito pré definido, tipicamente se adotam rubricas, que consistem em um conjunto de critérios de avaliação (Allen *et al.*, 2006) e os níveis de desempenho alcançáveis em cada critério (Biagiotti, 2005).

Para facilitar a avaliação no ensino de computação na educação básica é interessante automatizar a avaliação (Alves *et al*., 2019). A avaliação da aprendizagem dos alunos na área de computação e design de interface pode ser uma tarefa complexa e trabalhosa para os professores por diversos fatores. Um dos fatores é que muitos professores que ensinam informática não possuem formação para tal (De Luca et al., 2010) podendo ter dificuldades em realizar uma avaliação manual da forma correta. Outro benefício da avaliação automatizada é que ela proporciona imparcialidade.

Com esse objetivo, já foi criada a ferramenta web CodeMaster UI Design v1.0 (Solecki, 2020), que permite automaticamente avaliar o design de interface de aplicativos desenvolvidos com o App Inventor como resultado da aprendizagem, com base em uma rubrica. A ferramenta atualmente apresenta a pontuação analisando o design de interface de um app criado pelo aluno e, com base na rubrica, o aluno pode identificar em quais aspectos o design visual do seu aplicativo pode ser melhorado (Solecki, 2020). A rubrica utilizada no CodeMaster v1.0 foi também avaliada em termos da sua confiabilidade e validade demonstrando uma boa confiabilidade (sendo medida por meio do coeficiente alfa de Cronbach que alcançou um resultado satisfatório de $\alpha =$

0,84) e validade (com a maioria dos itens apresentando bons parâmetros de inclinação via Teoria de Resposta ao Item) (Solecki *et al*., 2020).

Porém, desde que este modelo de avaliação foi criado, os guias de estilo evoluíram e o Material Design passou por diversas atualizações. Assim, a proposta deste trabalho é que seja realizado um aprimoramento do modelo de avaliação, incluindo a rubrica e consequentemente a ferramenta CodeMaster UI Design v1.0, para incluir as evoluções dos guias de estilo e complementar a avaliação. Como resultado, espera-se fornecer uma ferramenta atualizada que traz auxílio aos professores e alunos da educação básica na avaliação do design de interface de projetos. Com isso espera-se facilitar ainda mais o ensino e popularização de computação e design de interfaces no Brasil.

1.2. OBJETIVOS

Objetivo geral

O objetivo deste trabalho é evoluir um modelo para automaticamente avaliar o design de interface de aplicativos móveis desenvolvidos com App Inventor no contexto da educação básica, acompanhando a evolução dos guias de estilo e complementando a avaliação com base na literatura. A avaliação é realizada com base na análise estática do código do projeto App Inventor, avaliando o grau de conformidade do design de interface com guias de estilo e métricas de qualidade de design de interface, e é guiada por uma rubrica.

Objetivos específicos

O1. Analisar a fundamentação teórica sobre ensino e aprendizagem do design de interface de usuário na educação básica; o desenvolvimento de design de interface com App Inventor e guias de estilo de design de interface de apps Android, e o CodeMaster UI Design V1.0.

O2. Analisar o estado da arte sobre modelos de avaliação (automatizada) de design de interface de apps criados com App Inventor no contexto da educação básica.

O3. Revisar e evoluir o modelo de avaliação CodeMaster - Design de UI (rubrica).

O4. Implementar e testar os ajustes da evolução do modelo de avaliação.

Premissas e restrições

O trabalho é realizado de acordo com o regulamento vigente do Departamento de Informática e Estatística (INE – UFSC) para os Trabalhos de Conclusão de Curso. A aprimoração a ser desenvolvida tem como foco a análise e avaliação de design de interface de apps Android, somente de projetos desenvolvidos com a ferramenta de desenvolvimento App Inventor no ensino de computação na educação básica. Este trabalho pretende tratar apenas dos conceitos de design de interface de aplicativos para dispositivos Android.

1.3. MÉTODO DE PESQUISA

A metodologia de pesquisa utilizada neste trabalho é dividida em quatro etapas.

Etapa 1 – Fundamentação teórica

Estudando, analisando e sintetizando os conceitos principais e a teoria referente aos temas a serem abordados neste trabalho é apresentada a fundamentação teórica. Nesta etapa são realizadas as seguintes atividades:

A1.1 – Análise teórica sobre ensino e aprendizagem de design de interface na educação básica

A1.2 – Análise teórica sobre o desenvolvimento de design de interfaces com App Inventor

A1.3 – Análise teórica sobre guias de estilo de design de interface de apps Android (Material Design).

A1.4 - Análise da arquitetura atual do CodeMaster UI Design V1.0

Etapa 2 – Estado da Arte

Nesta etapa é realizado um mapeamento sistemático seguindo o processo proposto por Petersen *et al*. (2015) para identificar e analisar modelos de avaliação de design de interface de apps com App Inventor atualmente sendo utilizados. Esta segunda etapa é dividida nas seguintes atividades:

A2.1 – Definição do protocolo da revisão:, search string, fontes, e os critérios de inclusão e exclusão

A2.2 – Execução da busca e seleção de artigos relevantes

A2.3 – Extração dos dados e análise realizando o mapeamento.

Etapa 3 – Evolução do modelo de avaliação

O modelo de avaliação de design de interface CodeMaster UI Design é melhorado com base na fundamentação teórica e do estado da arte.

A3.1 – Análise de problemas comuns de design de interfaces de apps criados com App Inventor

A3.2 – Evolução da rubrica de avaliação.

A3.3 - Avaliar a concordância da rubrica

Etapa 4 – Evolução da ferramenta automatizando a avaliação

Nesta etapa é evoluída a ferramenta CodeMaster UI Design ajustando a ferramenta de acordo com as mudanças na rubrica, com base no processo iterativo incremental de engenharia de software (Larman *et al*., 2003). Esta etapa é dividida nas seguintes atividades:

A4.1 – Análise de requisitos

A4.2 – Design de interface

A4.3 – Modelagem da arquitetura do sistema

A4.4 – Modelagem detalhada e implementação

A4.5 – Testes do sistema.

1.4 ESTRUTURA DO DOCUMENTO

O restante deste documento está estruturado da seguinte forma. No capítulo 2 é apresentada a fundamentação teórica sobre o ensino e aprendizagem de design de interface na educação básica, o App Inventor, os guias de estilo existentes e a ferramenta Code Master. No capítulo 3 um mapeamento sistemático de abordagens existentes para a avaliação da interface de usuário de aplicativos é apresentado. No capítulo 4 é apresentada uma sugestão de evolução da rubrica do Code Master UI Design, para avaliação de design visual de aplicativos. No capítulo 5, é feita uma descrição da implementação realizada. No capítulo 6, é realizada uma avaliação da concordância do modelo de avaliação e são discutidos os resultados. No capítulo 7 são apresentados os resultados alcançados e sugestões de trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica que sustenta o desenvolvimento da rubrica, explorando conceitos e práticas relevantes para o trabalho.

2.1 ENSINO E APRENDIZAGEM DE DESIGN DE INTERFACE NA EDUCAÇÃO BÁSICA

O foco de unidades instrucionais para o ensino de computação na educação básica tipicamente está direcionado ao conceito de Algoritmos e Programação, entre outros conceitos típicos da computação como redes e impactos da computação (*Computer Science Teachers Association*, 2016) (Sociedade Brasileira de Computação, 2018). No entanto, outro conceito importante é o conceito transversal Interação Humano-Computador, que é o estudo de como as pessoas interagem com computadores e até que ponto os sistemas de computação são ou não são desenvolvidos para uma interação bem sucedida com seres humanos (*Computer Science Teachers Association*, 2016). Sendo assim, existem vantagens para os alunos com a complementação da aprendizagem de computação na escola com o ensino de design de interface. Uma delas é aprender as competências básicas de design gráfico que permite que os estudantes se tornem comunicadores mais confiantes (*American Institute of Graphic Arts*, 2022).

A AIGA - *The professional association for design* propõe um currículo que define conteúdos importantes para o ensino de design de interface no ensino médio (*American Institute of Graphic Arts*, 2022). Este currículo acompanha os estudantes por meio do processo de design e dá-lhes práticas de resolução de problemas a fim de aumentar as suas competências como designers gráficos. O currículo possui quatro unidades conforme detalhado na Tabela 1.

Tabela 1. Unidades e os objetivos de aprendizagem relacionada ao design gráfico (*American Institute of Graphic Arts,* 2022)

Unidade	**Objetivos de aprendizagem**
Introdução ao design gráfico	• Gerar várias ideias no formulário de desenhos em miniatura • Determinar quais de suas ideias são as melhores • Fazer rascunhos • Fazer uma composição final
Introdução a noções básicas de design 2D	• Desenvolver uma consciência do papel formal que os pontos, linhas e planos desempenham na arte e no design por meio do uso de recursos visuais e

	discussões em classe • Desenvolver uma consciência do papel formal da teoria da Gestalt na criação de arte e design através do uso de recursos visuais e discussão em classe • Desenvolver uma consciência das propriedades físicas da cor e do papel expressivo da cor na cultura visual através do uso de recursos visuais e discussão em classe • Prepará-los para se envolver e ter sucesso nas atividades e atribuições da Unidade 2 B–D
Introdução ao processo de design	Os alunos vão… • Analisar e entender as conexões fortes entre arte e design • Perceber que o processo de design é apenas isso - um processo • Começar a entender as diferenças entre design eficaz e ineficaz
Introdução à tipografia	Os alunos vão… • Analisar o papel da tipografia na vida cotidiana • Explicar os efeitos da tipografia funcional • Identificar experiências ou produtos que precisam de reformulação tipográfica

Para cada unidade o guia de currículo da AIGA (2022) propõe rubricas com o objetivo de facilitar a avaliação da aprendizagem do aluno.

Tabela 2. Rubrica da unidade 1 referente a avaliação da aprendizagem da Introdução ao design gráfico (*American Institute of Graphic Arts*, 2022)

Critérios/Tarefa	**Níveis de desempenho**		
	Iniciante	**Em desenvolvimento**	**Competente**
	6-9 pontos	10-12 pontos	13-15 pontos
O aluno demonstra consciência de quais são os elementos do design - linha, forma, textura e cor (dependendo na tarefa) e entende como usá-los com variedade e intenção. O aluno inventou 16 diferentes ideias de miniaturas para exercícios.	O aluno usa o elemento conforme atribuído, mas mostra pouca ou nenhuma variedade de exploração. O aluno foi capaz de expandir a variedade - algumas miniaturas, mas lutou para chegar a mais ideias. O aluno conseguiu com mais informações e orientações. Enviou poucos exemplos.	O aluno usa o elemento conforme atribuído, e mostra alguma variedade de exploração, mas tende a confiar em exemplos familiares, seguros ou repetitivos. O aluno foi capaz de expandir variedade em thumbnails, e, principalmente. exibiu intenção e um senso claro de exploração sem muita repetição ou assistência. Pode ter enviado também alguns exemplos.	O aluno usa o elemento conforme atribuído, e mostra grande variedade e exploração. O aluno foi capaz de expandir a variedade em 16 miniaturas e mostrar intenção e um senso claro de exploração sem muita repetição. O estudante inclina-se independente para solucionar problemas.
	1-2 pontos	3 pontos	4-5 pontos
O aluno usa análise e intenção para avaliar quais miniaturas são seus três melhores conceitos para esboçar.	O aluno foi capaz de selecionar três miniaturas para esboçar depois da orientação, assistência direta e revisitar o que funciona e por que quanto à composição.	O aluno foi capaz de diferenciar e selecionar algumas miniaturas para esboçar, mas precisava de um pouco de ajuda para tomar a decisão final.	O aluno foi capaz de discriminar e selecionar os três melhores para esboçar com base em intenção clara, compreensão e análise do que funciona e por quê. O estudante se direciona para solucionar problemas de forma independente.
	6-9	10-12	13-15
O estudante é capaz de ajudar a escolher um rascunho para converter em um projeto mais desenvolvido e é capaz de resolvê-lo para uma imagem acabada com marcador negro.	O estudante auxilia na escolha de um rascunho para se converter em um projeto. O aluno completa um projeto de imagem com base no rascunho selecionado. A imagem final parece um pouco incompleta ou possivelmente feita apressadamente.	O estudante auxilia na escolha de um rascunho para se converter em um projeto. O aluno completa um projeto de imagem com base no rascunho selecionado. A imagem final parece mais completa ou talvez precisasse de mais tempo ou atenção ao artesanato. O estudante inclina-se principalmente para a solução independente de problemas.	O estudante ou escolhe ou é capaz de ajudar na escolha de um rascunho para converter em um projeto mais desenvolvido. O estudante completa uma imagem acabada com base no rascunho selecionado. A imagem final parece completa, resolvida, intencional, e cuidada. O estudante inclina-se para solução independente de problemas.

Tabela 3. Rubrica: Hierarquia e emparelhamento de fontes referente a avaliação da aprendizagem da Introdução à tipografia (*American Institute of Graphic Arts*, 2022)

Poemas tipográficos para livro de poesia de classe									
	8	7	6	5	4	3	2	1	0
	Missão superada		Cumpriu a tarefa		A maioria cumpriu a tarefa		Dificilmente cumpriu a tarefa		Não tentou

Cinco poemas foram selecionados									
Emparelhamento proposital de fontes para melhorar significado									
Tentou mais de uma versão de cada poema									
Criou um visual hierárquico que melhora o significado do poema									
Correção ortográfica									
Notas									

Considerando a importância do ensino de design visual na educação básica, essas diretrizes de currículo guiam o ensino de design na educação básica propondo os temas relevantes a serem trabalhados e o que deve ser exercitado. Analisando o *Computer Science Teachers Association* (*Computer Science Teachers Association*) (2016) observa-se que as principais áreas incluem sistemas de computação, algoritmos e programação, dados e análise, redes e internet e impactos da computação. Mais relacionado com a área de interação humano computador são as áreas de Sistemas de computação e Algoritmos e programação, detalhadas na Tabela 4.

Tabela 4. Currículo de computação para as áreas de Sistemas de computação e Algoritmos e programação (*Computer Science Teachers Association*, 2016)

	Sistemas de computação	**Algoritmos e programação**
Até grade 2	Dispositivos: uso de dispositivos de computação Hardware e software: compõem o sistema de computação Solução de problemas: descrição clara de um problema é o primeiro passo	Programação: pessoas desenvolvem programas de forma colaborativa para um propósito Variáveis: o tipo de dados determina as ações e atributos Controle: computador segue sequência de instruções Modularidade: tarefas complexas podem ser divididas Algoritmos: processos podem ser expressos em algoritmos
Até final da grade 5	Dispositivos: podem ser conectados a outros dispositivos ou componentes Hardware e software: trabalham em conjunto como um sistema para realizar tarefas, tais como como envio, recebimento, processamento e armazenamento de unidades de informação Solução de problemas: estratégias comuns de solução de problemas são eficazes em muitos sistemas	Programação: pessoas desenvolvem programas utilizando um processo iterativo envolvendo o projeto, implementação, e revisão. Variáveis: são utilizadas para armazenar e modificar dados Controle: estruturas de controle, incluindo loops, manipuladores de eventos, e condicionadores, são usado para especificar o fluxo de execução. Modularidade: programas podem ser divididos em partes menores para facilitar seu projeto Algoritmos: diferentes algoritmos podem alcançar o mesmo resultado.

Até final da grade 8	Dispositivos: a interação entre humanos e dispositivos computacionais apresenta vantagens, desvantagens e consequências não intencionais. Hardware e software: determinam a capacidade de um sistema de computação de armazenar e informações sobre o processo. Solução de problemas: requer o conhecimento de como a computação, dispositivos e componentes funcionam e interagem.	Programação: as pessoas projetam soluções significativas para os outras Variáveis: os programadores criam variáveis para armazenar valores de dados de tipos selecionados. Controle: programadores selecionam e combinam estruturas de controle, tais como loops, eventos e condicionadores Modularidade: os programas utilizam procedimentos para organizar o código, ocultar detalhes de implementação, e facilitar a reutilização do código. Algoritmos: as pessoas projetam algoritmos que são generalizáveis a muitas situações.
Até final da grade 12	Dispositivos: os dispositivos computacionais são freqüentemente integrados com outros sistemas, incluindo sistemas biológicos, mecânicos e sociais Hardware e software: existem níveis de interação entre o hardware, o software e o usuário de um sistema informático. Solução de problemas: solução de problemas complexos envolve o uso de múltiplas fontes ao pesquisar, avaliar e implementar soluções potenciais.	Programação: diversas equipes podem desenvolver programas com amplo impacto através de revisão e aproveitando os pontos fortes dos membros em diferentes funções Variáveis: as estruturas de dados são usadas para gerenciar a complexidade do programa. Controle: os programadores consideram as compensações relacionadas à implementação, legibilidade e desempenho do programa ao selecionar e combinar estruturas de controle. Modularidade: os programas complexos são projetados como sistemas de módulos de interação Algoritmos: as pessoas avaliam e selecionam algoritmos com base no desempenho, reusabilidade, e facilidade de implementação.

Existem poucas iniciativas para ensinar o design de interface no ensino fundamental e médio. Alguns países, como por exemplo a China e a Coréia do Sul, já incluíram o ensino de design para os alunos de educação básica (West-Knights, 2017) (Ahn, 2012). Em outros lugares, também existe um movimento para ensinar design em escolas como por exemplo projeto City[3] ou o laboratório K-12 da Stanford d.school[4]. No Brasil, a fim de explorar a questão do ensino de design de UI na Educação Básica, foi desenvolvida uma unidade instrucional “Faça seu próprio app”, com o objetivo de ensinar competências básicas de design visual de UI, incorporada no ensino de computação por meio do desenvolvimento de aplicações móveis (Ferreira *et al*., 2020).

2.2 DESENVOLVIMENTO DE DESIGN DE INTERFACES COM APP INVENTOR

O App Inventor (*Massachusetts Institute of Technology*, 2019) é um ambiente de programação baseado em blocos utilizado para criar aplicações Android e iOS. Ele pode ser gratuitamente acessado por um navegador web. Ele permite tanto a parte da programação funcional quanto também o design de interface de usuário.

No processo de criação dos aplicativos, inicialmente os componentes da

[3] www.cityxproject.com
[4] https://dschool.stanford.edu/programs/k12-lab-network

interface de usuário (*User Interface - UI*) devem ser posicionados. Esses componentes são os elementos visíveis de UI do app inventor. Posteriormente as características destes componentes podem ser configuradas. Para usar um componente no aplicativo, é necessário clicar e arrastá-lo para o visualizador no meio do Designer. Quando é adicionado um componente ao Visualizador, ele também aparecerá na lista de componentes à direita do Visualizador (*Massachusetts Institute of Technology*, 2023). Os componentes têm propriedades ajustáveis. Essas propriedades alteram a maneira como o componente aparece ou se comporta no aplicativo. Para visualizar e alterar as propriedades de um componente, é selecionado o componente desejado em sua lista de componentes (*Massachusetts Institute of Technology*, 2023).

Figura 1. Visualizador, os Componentes e as propriedades (*Massachusetts Institute of Technology*, 2023)

O App Inventor (*core*) fornece diversos elementos visíveis de interface de usuário conforme listado na Tabela 5.

Tabela 5. Elementos visíveis do App Inventor

Elemento Visível	**Descrição**	**Exemplo**
Imagem de fundo	Uma imagem que aparece na parte de trás da tela.	

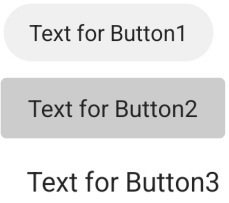
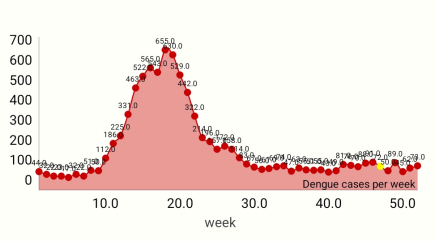
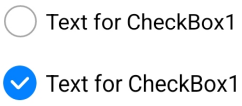
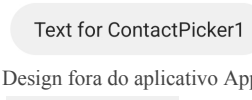
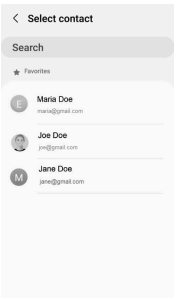
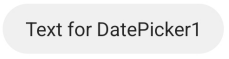
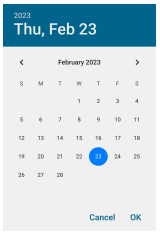
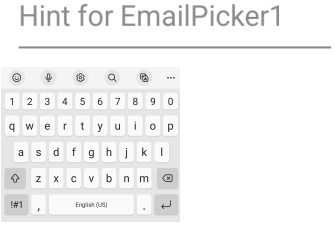
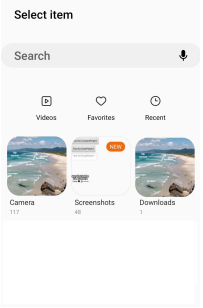
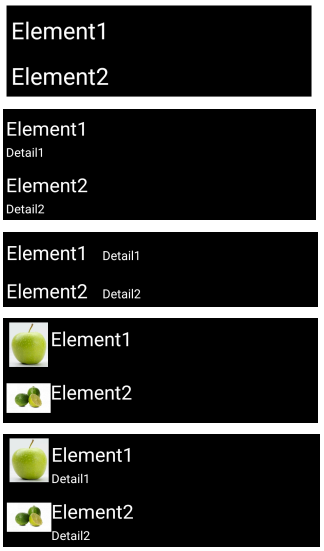
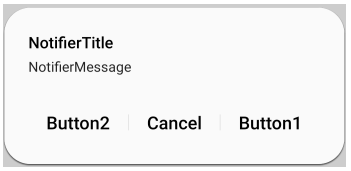
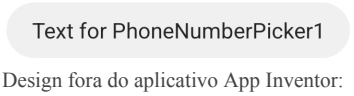
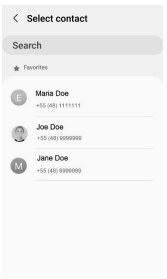
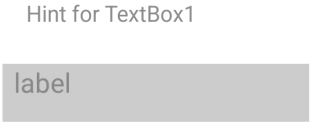
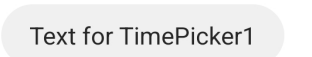
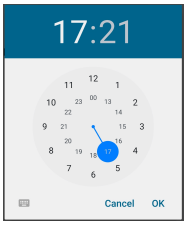
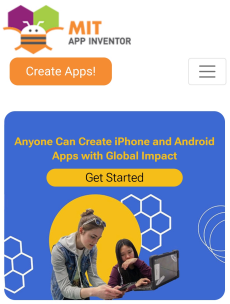
Botão	Um componente com a capacidade de detectar cliques: padrão, arredondado, retangular	 (também pode ser uma imagem, ícone)
Gráfico	Um componente que plota como linha, área, dispersão, barra e gráfico de setores	
Caixa de seleção	Um componente que gera um evento quando o usuário clica.	
Seletor de contato	Um botão que, ao ser clicado, exibe uma lista de contatos a serem escolhidos.	 Design fora do aplicativo App Inventor: 
Seletor de data	Um botão que, quando clicado, abre uma caixa de diálogo pop-up para permitir que o usuário selecione uma data.	 Design fora do aplicativo App Inventor: 
Seletor de email	Uma espécie de caixa de texto em que o usuário pode inserir o nome ou endereço de e-mail de um contato e o telefone mostrará um menu suspenso de opções que completam a entrada.	

Imagem	Um componente para exibir imagens.	
Seletor de imagem	Um botão de propósito especial. Quando o usuário toca em um seletor de imagens, a galeria de imagens do dispositivo aparece e o usuário pode escolher uma imagem.	Text for ImagePicker1 Design fora do aplicativo App Inventor: 
Legenda	Um componente para exibir um pedaço de texto, que é especificado por meio da propriedade Text.	Text for Label1
Seletor de lista	Um botão que, ao ser clicado, exibe uma lista de textos para o usuário escolher.	Text for ListPicker1 
Exibidor de lista	Um componente para exibir uma lista de elementos de texto e imagem	
Mapa	Um contêiner bidimensional que renderiza blocos de mapa em segundo plano e permite que vários elementos Marcadores identifiquem pontos no mapa.	

Notificador	Um componente que exibe caixas de diálogo de alerta, mensagens e alertas temporários.	 (Várias versões do mesmo pop-up com menos botões)
Caixa de senha	Uma caixa de texto para inserir senhas.	
Seletor de número de telefone	Um botão que, ao ser clicado, exibe uma lista com os números de telefone dos contatos a serem escolhidos.	 Design fora do aplicativo App Inventor: 
Controle deslizante	Uma barra de progresso que adiciona uma miniatura arrastável.	
Spinner	Um componente que exibe um pop-up com uma lista de elementos.	 
Interruptor	Um componente que gera um evento quando o usuário clica nele.	
Caixa de texto	Uma caixa para o usuário inserir texto.	 *Este elemento não tem uma borda*
Seletor de horário	Um botão que, quando clicado, abre uma caixa de diálogo pop-up para permitir que o usuário selecione um horário.	 Design fora do aplicativo App Inventor: 

Reprodutor de vídeo	Um componente multimídia capaz de reproduzir vídeos.	
Visualizador Web	Um componente para visualizar páginas da Web.	Shows any website: 

Para personalizar a aparência dos componentes da interface do usuário em seu aplicativo, cada componente possui propriedades que podem ser modificadas. Algumas propriedades são cor, tamanho, alinhamento e tipografia. Cada componente possui propriedades distintas. Por exemplo, o componente Botão possui algumas propriedades como altura, largura, forma, cor de fundo, cor de texto, tamanho de fonte, etc.

2.3 GUIAS DE ESTILO DE DESIGN DE INTERFACE DE APPS ANDROID

Com o propósito de padronizar os princípios de design visual e auxiliar na aplicação de determinadas diretrizes relacionadas ao design de interfaces, surgiram os guias de estilo que facilitam a implementação de interfaces com usabilidade e acessibilidade.

O principal exemplo de guia de estilo para aplicativos Android é o Material Design (GOOGLE, 2022) que possibilita a consistência visual entre aplicativos da mesma plataforma. O Material Design foi originalmente criado em 2014 e atualmente está sendo disponível na versão 3 (MD3). O Material Design contém diretrizes sobre a aparência dos componentes da interface e como eles devem reagir a interações do usuário. Ele fornece diretrizes específicas para diversos componentes e aborda aspectos como cor, elevação, ícones, movimento, forma e tipografia.

O sistema de design é organizado em três principais partes: fundamentos e customização, estilos, e componentes. Os fundamentos e customização definem conceitos de acessibilidade, layout, estados de interação e tokens de design. Estilos são

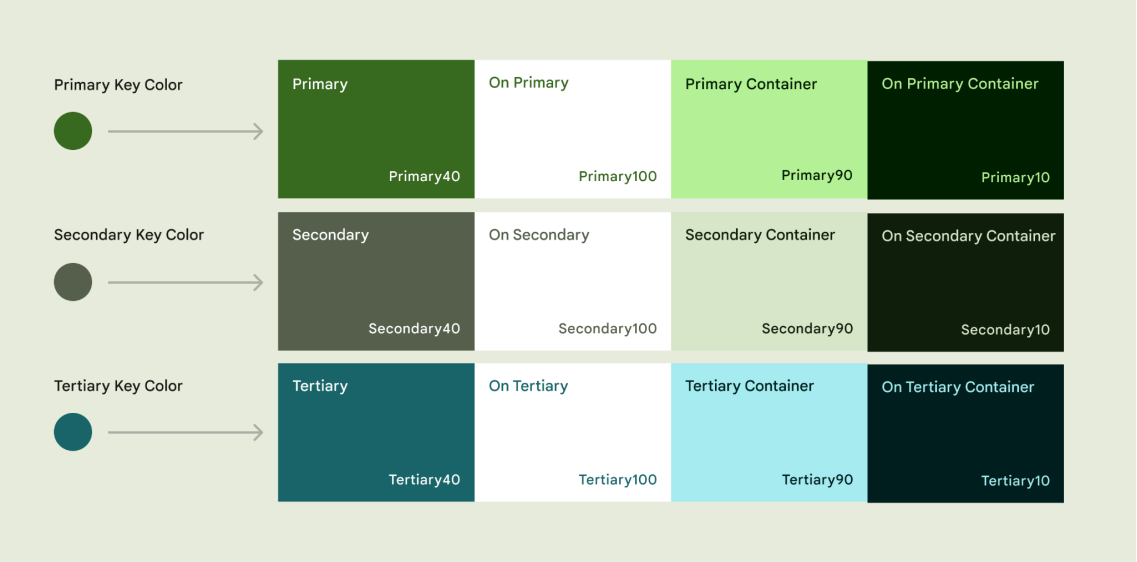
sistemas de aplicação de cor, tipografia, ícones, movimento e forma. Os componentes se referem a elementos customizáveis.

2.3.1 Cor

A cor é usada para expressar estilo e comunicar significado. O Material Design 3 define um esquema de cores que tem como base um conjunto de cinco cores-chave que se relacionam individualmente com paletas tonais separadas com 13 tons. Tons específicos de cada paleta tonal são atribuídos a funções de cores em uma interface do usuário. As cores principais são a base para a criação de qualquer esquema de cores dinâmico. Com as cores-chave estabelecidas, o guia do Material Design especifica todo o espectro de cores necessário para expressar estados de interação, erros e contraste acessível.

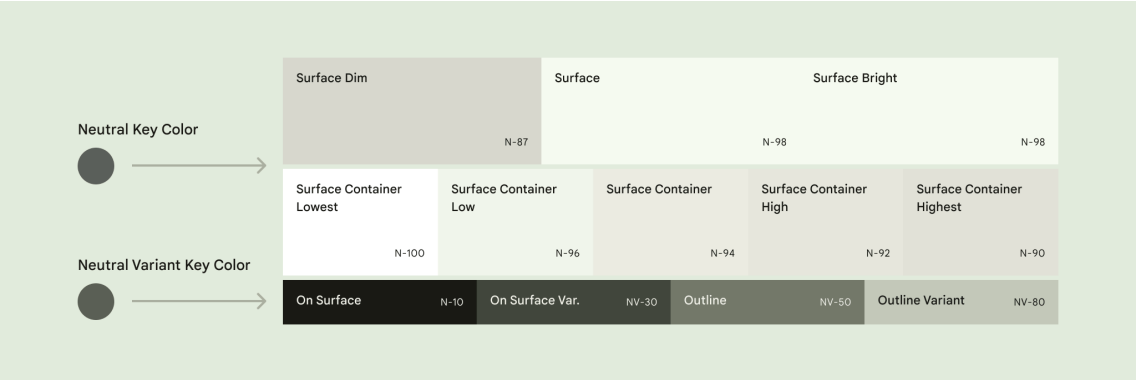
A cor-chave primária é usada para derivar funções para os principais componentes da interface do usuário, como o botão de ação flutuante (FAB), botões proeminentes, estados ativos, bem como a tonalidade das superfícies elevadas. A cor-chave secundária é usada para componentes menos proeminentes na interface do usuário, como chips de filtro, enquanto expande a oportunidade de expressão de cor. A cor-chave terciária é usada para derivar os papéis dos acentos contrastantes que podem ser usados para equilibrar as cores primárias e secundárias ou chamar a atenção para um elemento. A função da cor terciária é deixada para as equipes usarem a seu critério e destina-se a apoiar uma expressão de cor mais ampla nos produtos.

Figura 2. Cores-chave primárias, secundárias e terciárias (Google, 2023).



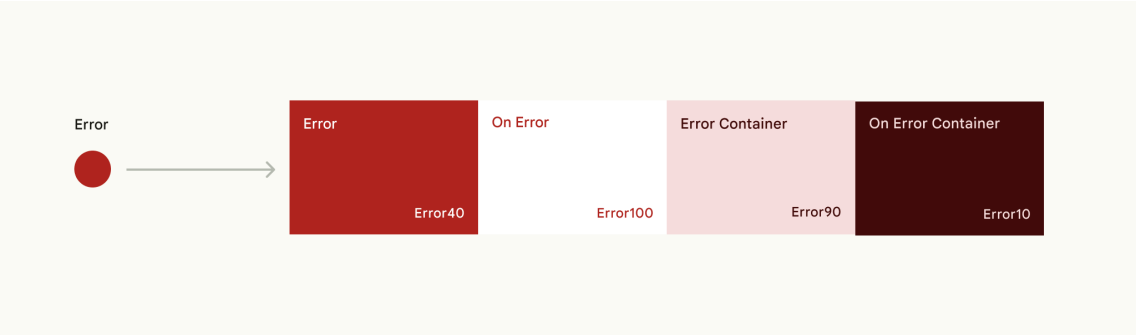
A cor-chave neutra é usada para derivar papéis de cores de superfície para planos de fundo, bem como cores usadas para texto e ícones de alta ênfase. A cor-chave variante neutra é usada para derivar funções de cores para elementos de ênfase média, como texto, ícones e contornos de componentes.

Figura 3. Cor chave neutra e cor chave variante neutra (Google, 2023)



Além do acento e da cor-chave neutra, o sistema de cores inclui uma função de cor semântica para erro, novamente na forma da própria função de erro, além de uma função on-erro , erro container e on-erro container.

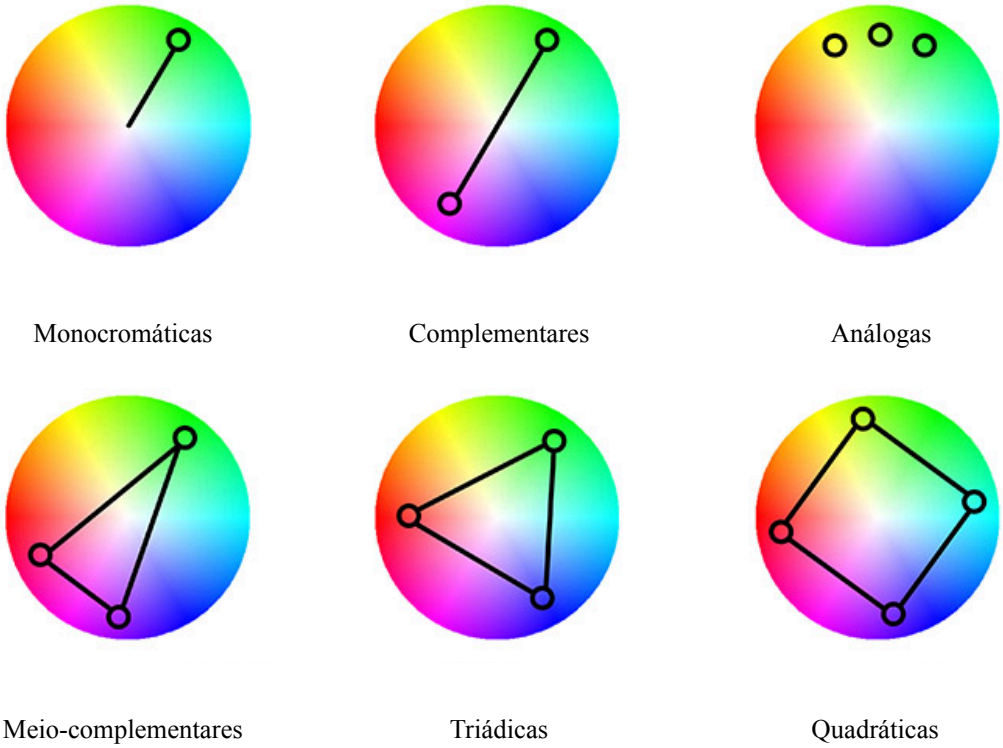
Figura 4. Cores de erro (Google, 2023)



Um aspecto importante do esquema de cores utilizado na interface é 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 é o RYB (Color Matters, 2019a), que considera vermelho, amarelo e azul como cores primárias. 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.

Com a mistura das cores primárias, formam-se as cores intermediárias. As cores primárias e as cores intermediárias dispostas em círculo formam o círculo cromático. A posição das cores no círculo cromático define as relações de harmonia entre as cores. O círculo cromático indica as combinações ideais. Algumas relações básicas de harmonia são cores complementares, triangulação e análogas. Cores complementares são as que no círculo cromático estão posicionadas nas extremidades opostas. Cores monocromáticas são compostas de uma tonalidade única. Na combinação de triangulação, as cores formam um triângulo equilátero no círculo. As cores quadráticas são as que formam um quadrado no círculo cromático. As cores meio-complementares ocorrem quando ao invés de se optar pela cor oposta, são escolhidas as cores vizinhas. Já as cores análogas estão próximas umas das outras.

Figura 5. Cores complementares, triangulação e análogas.[5]



Existem algumas propriedades usadas para descrever as cores, como matiz, saturação e luminosidade. Matiz refere-se à própria cor pura, como vermelho, verde ou azul. A saturação refere-se à intensidade ou pureza da tonalidade. Uma cor altamente saturada é muito viva e parece mais pura, enquanto uma cor menos saturada parece mais desbotada ou suave. A luminosidade refere-se à quantidade percebida de luz em uma cor. As cores mais claras têm um valor ou brilho mais alto, enquanto as cores mais escuras têm um valor ou brilho mais baixo.

Os sistemas de cores que melhor representam como os humanos percebem as cores são o HSL (*Hue, Saturation, Lightness*) e o HCL (*Hue, Chroma, Lightness*). No formato HSL a matiz recebe um número entre 0 e 360, a saturação recebe um número entre 0 e 100% e a luminosidade recebe um número entre 0 e 100%. No formato HCL a matiz recebe um número entre 0 e 360, o chroma recebe um número entre 0 e 131 e a luminosidade recebe um número entre 0 e 100%. Este formato também permite fácil verificação de contraste.

[5] https://www.des1gnon.com/2017/11/usar-contraste-de-cor/

2.3.2 Tipografia

A tipografia define o formato dos caracteres e letras. Uma família tipográfica é composta por diversas fontes, que são variações de um mesmo formato básico. A família tipográfica padrão do Android é o Roboto (Google, 2023) e é do tipo sem serifa.

Para que os usuários consigam identificar as funcionalidades do aplicativo com mais facilidade, deve-se limitar a quantidade 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. Cada um desses estilos possui um tamanho de fonte definido.

Um atributo principal é a espessura do traço. A espessura mais comum é regular, porém pode abranger extremos desde muito finos a muito grossos, como por exemplo Bold (utilizado para dar ênfase).

Figura 6. Conjunto de estilos predefinidos conforme papel no texto (Google, 2023)

https://m3.material.io/styles/typography/type-scale-tokens

2.3.4 Ícones

Os ícones podem ser usados para representar ações comuns. Símbolos de materiais são um conjunto de fontes de ícones variáveis criadas em sete pesos em três estilos diferentes.

O conjunto de fontes de ícones variáveis fornecido pelo MD3 oferece suporte a três estilos: contornado, arredondado e nítido.

Figura 7. Exemplos de ícones de estilos contornado (1), arredondado (2) e nítido (3) (Google, 2023)

É importante no design de interface usar ícones da mesma família e do mesmo estilo, pois a coesão visual facilita que a mensagem seja passada de forma clara, intuitiva e estéticamente agradável. Nos ícones em cores, é importante que eles sigam a paleta de cores do design de interface, promovendo uma experiência de usuário mais positiva. Essa coesão visual está em consonância com diretrizes do Material Design.

2.3.5 Imagens

Imagens tem diversos usos em um app, como representar exemplos ou ações. A qualidade das imagens contribui para que a mensagem seja transmitida efetivamente. Existem requisitos mínimos quanto a sua qualidade. Um desses requisitos é a resolução, que a partir de 72 dpi (pontos por polegada) pode ser considerada satisfatória. Ampliar uma imagem demais pode deixá-la pixelizada ou embaçada, o que prejudica sua

qualidade. Outro requisito de qualidade da imagem é a sua proporção. Para usar uma imagem com uma proporção diferente da original, ela deve ser cortada, e não distorcida. Outro fator importante que determina a qualidade da imagem, é se ela possui fundos brancos que comprometam a estética da aplicação.

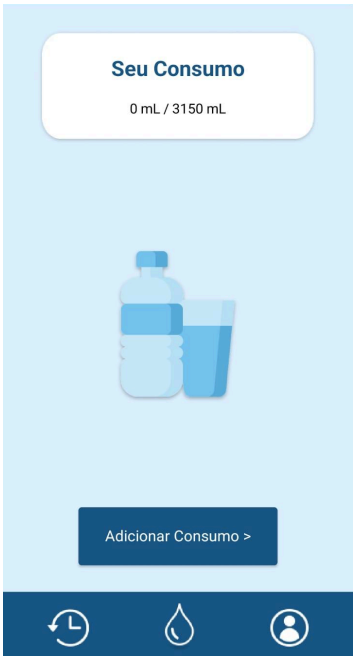


Figura 8a. Exemplo de tela com imagem com proporção e qualidade adequada

Figura 8b. Exemplo de tela com imagem com proporção distorcida

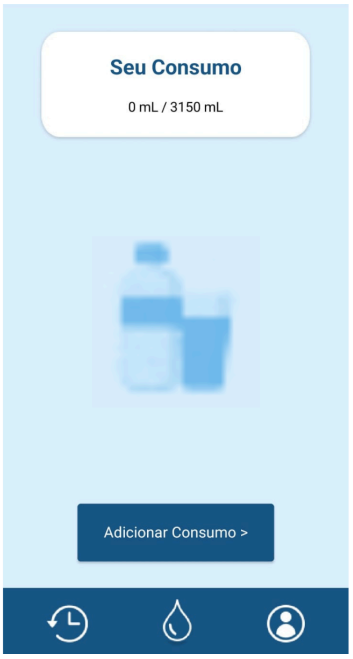
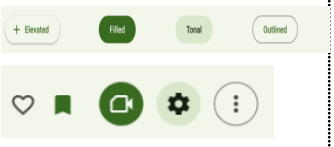


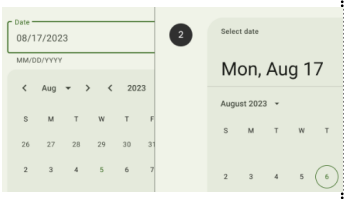
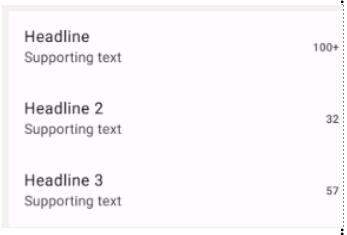
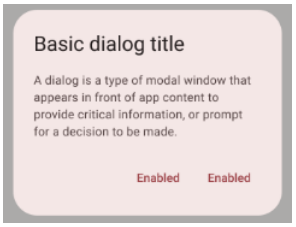
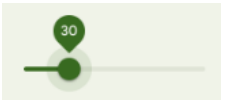
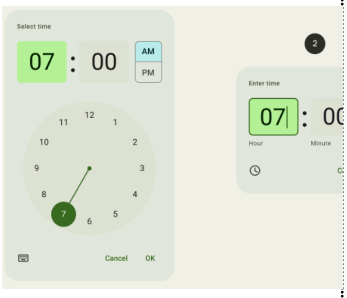
Figura 8c. Exemplo de tela com imagem embaçada

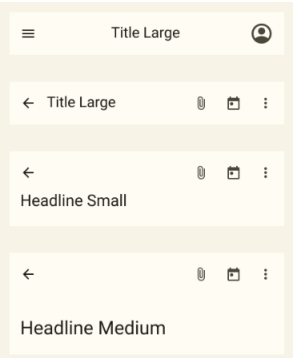
2.3.6 Componentes

O MD3 também apresenta diretrizes para diversos componentes de interface incluindo tanto o formato, fontes, e tamanhos adequados conforme Tabela 6.

Tabela 6. Exemplos de diretrizes para componentes de interface

Material Design 3 componente	**Breve descrição**	**Exemplo**
Botão	Permite a detecção de cliques	

Caixa de seleção	Permitem que os usuários selecionem um ou mais itens de um conjunto. Podem ativar ou desativar uma opção.	
Selecionador de data	Permitem que as pessoas selecionem uma data ou um intervalo de datas	
Listas	Índices verticais contínuos de texto ou imagens	
Diálogo	Fornecem prompts importantes em um fluxo de usuário	
Controle deslizante	Permite que os usuários façam seleções a partir de uma gama de valores	
Interruptor	Ativa ou desativa o estado de um único item	
Seletor de horário	Ajuda os usuários a selecionar e definir um horário específico	

Barra de aplicativo superior	Exibe informações e ações na parte superior da tela	

Para cada componente o MD3 também fornece uma especificação detalhada do container. Por exemplo para o botão, existem regras para o container, texto e ícones, que definem sugestões de algumas características como cores e tamanhos conforme ilustrado na Figura 9 e Figura 10.

Figura 9. Especificação para botão do tipo elevado (Google, 2023).

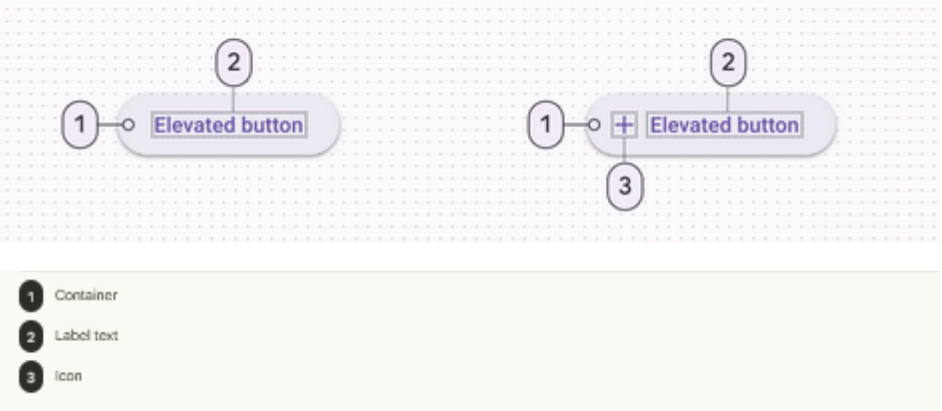
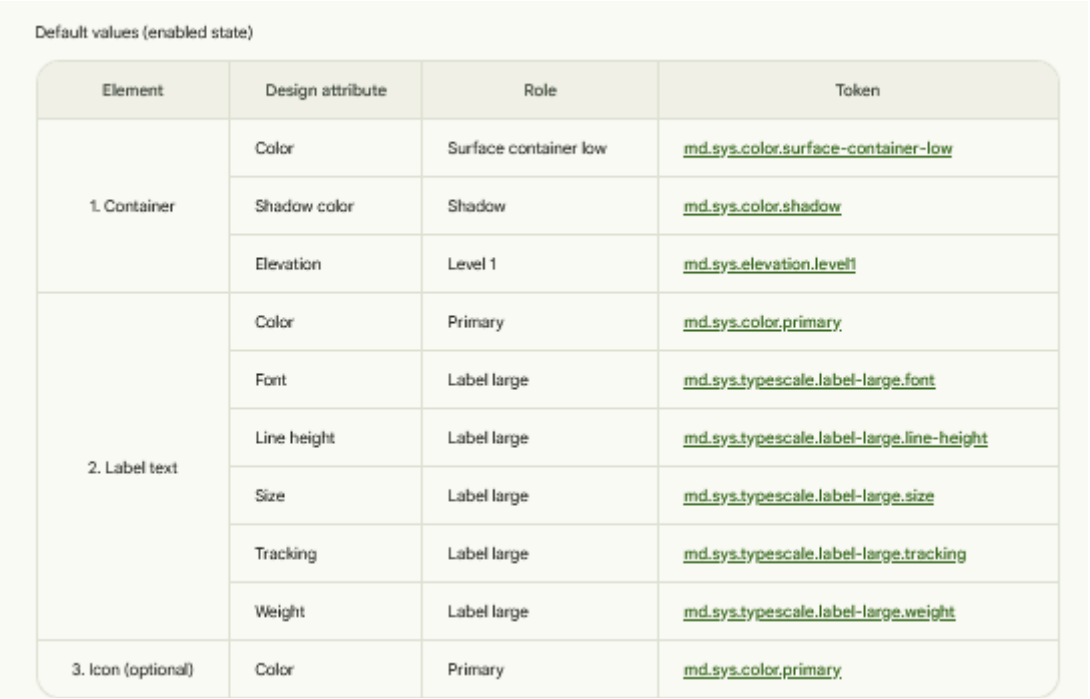


Figura 10. Especificações para botão do tipo elevado em valor default (Google, 2023).

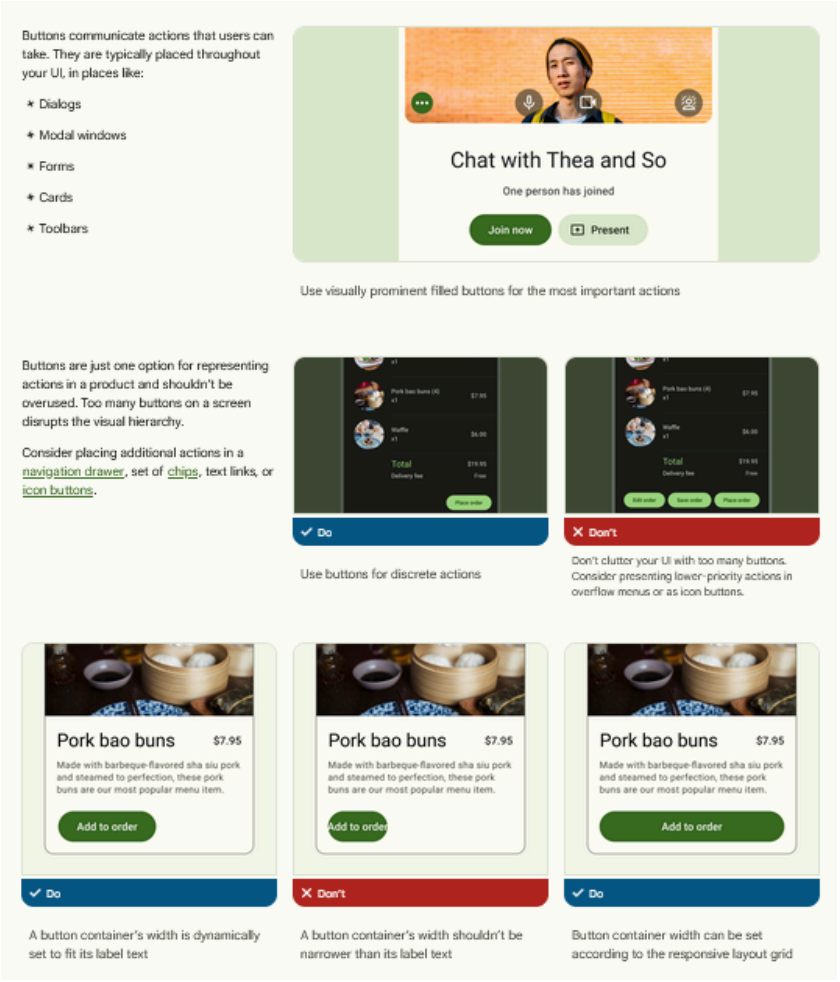


Default values (enabled state)

Element	Design attribute	Role	Token
1. Container	Color	Surface container low	md.sys.color.surface-container-low
	Shadow color	Shadow	md.sys.color.shadow
	Elevation	Level 1	md.sys.elevation.level1
2. Label text	Color	Primary	md.sys.color.primary
	Font	Label large	md.sys.typescale.label-large.font
	Line height	Label large	md.sys.typescale.label-large.line-height
	Size	Label large	md.sys.typescale.label-large.size
	Tracking	Label large	md.sys.typescale.label-large.tracking
	Weight	Label large	md.sys.typescale.label-large.weight
3. Icon (optional)	Color	Primary	md.sys.color.primary

Além disso, o Material Design fornece diretrizes sobre quando é adequado usar determinado componente mostrando *how to do's* e *how not to do's* conforme mostrado na Figura 11.

Figura 11. Diretrizes detalhadas para uso de botões (Google, 2023)



2.4 CODEMASTER

2.4.1 Modelo de avaliação do CodeMaster - design de interface

O projeto CodeMaster v1.0[6] (Computação na Escola, 2023) é uma aplicação web *full-stack* que permite avaliar projetos do MIT App Inventor automaticamente. Atualmente o sistema avalia conceitos de algoritmos de programação (Alves *et. al.*, 2020) e também conceitos de design visual de aplicativos móveis em conformidade com o Material Design (Solecki, 2019) e criatividade (Alves *et al.*, 2023).

Os principais aspectos que são avaliados no quesito de design visual no CodeMaster englobam elementos como *layout*, cor, tipografia e imagens de acordo com meta-princípios de design (Garret, 2011) (Schlatter *et al.*, 2013) como consistência, hierarquia e personalidade. O objetivo é fazer o uso balanceado dos elementos de design

[6] http://apps.computacaonaescola.ufsc.br/codemaster/

seguindo os meta-princípios para resultar em uma interface bonita e amigável.

A avaliação do design de interface de usuário de projetos criados com o App Inventor do CodeMaster é baseada em uma rubrica. A versão atual da rubrica v1.0 (Solecki, 2019) leva em consideração o *layout*, tipografia, cores e imagens em conformidade com o Material Design 1 para analisar a conformidade do design de interface de app criado pelo estudante com App Inventor no contexto da educação básica. Ela define 2-3 níveis de desempenho e atribui uma pontuação para cada critério conforme apresentado representado na Tabela 7.

Tabela 7. Rubrica CodeMaster UI Design – App Inventor v1.0 (SOLECKI, 2019)

Critério	**0pt**	**1pt**	**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 ≥ 20	min ≥ 2 e max ∈ [10, 19]	min ≥ 2 e max ≤ 9
Tipografia			
T1. Todos os componentes usam a 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 fontes recomendadas 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

A rubrica foi avaliada em termos de confiabilidade e validade. Como resultado apresentou boa confiabilidade, conforme o valor do coeficiente alfa de Cronbach ($\alpha = 0,84$) para medir a consistência da rubrica.

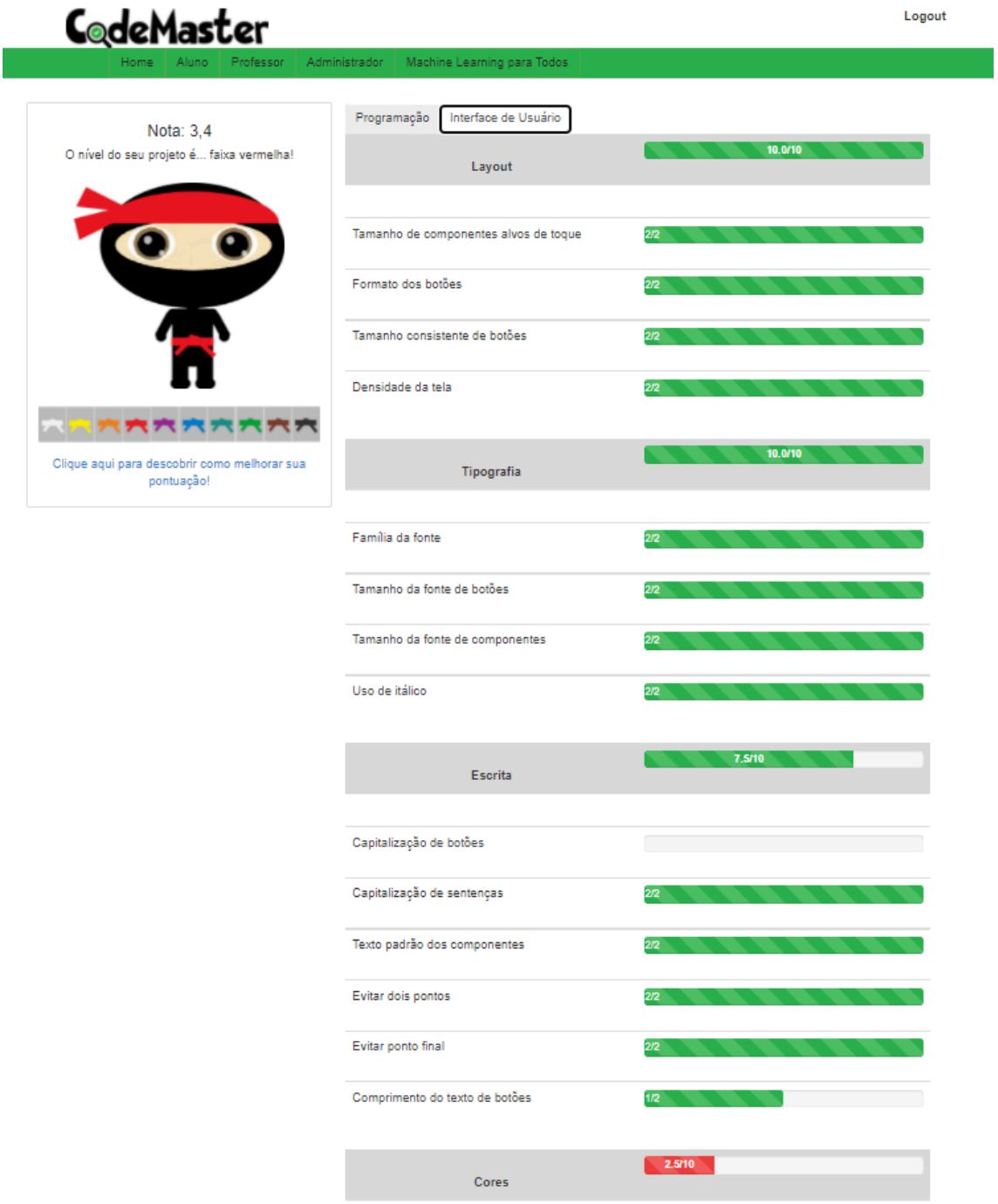
A avaliação com base nesta rubrica foi automatizada na ferramenta CodeMaster. Para iniciar a avaliação o usuário, utiliza-se o menu Aluno para fazer *upload* do arquivo App Inventor (.aia) referente ao projeto a ser avaliado, representado na Figura 12.

Figura 12. Upload de projeto App Inventor no CodeMaster[7].

Em seguida, o usuário é redirecionado para a tela de resultado da avaliação apresentando as pontuações, nota final e faixa de ninja com o objetivo de apresentar o resultado da avaliação de forma lúdica neste contexto da educação básica.

[7] http://apps.computacaonaescola.ufsc.br/codemaster/

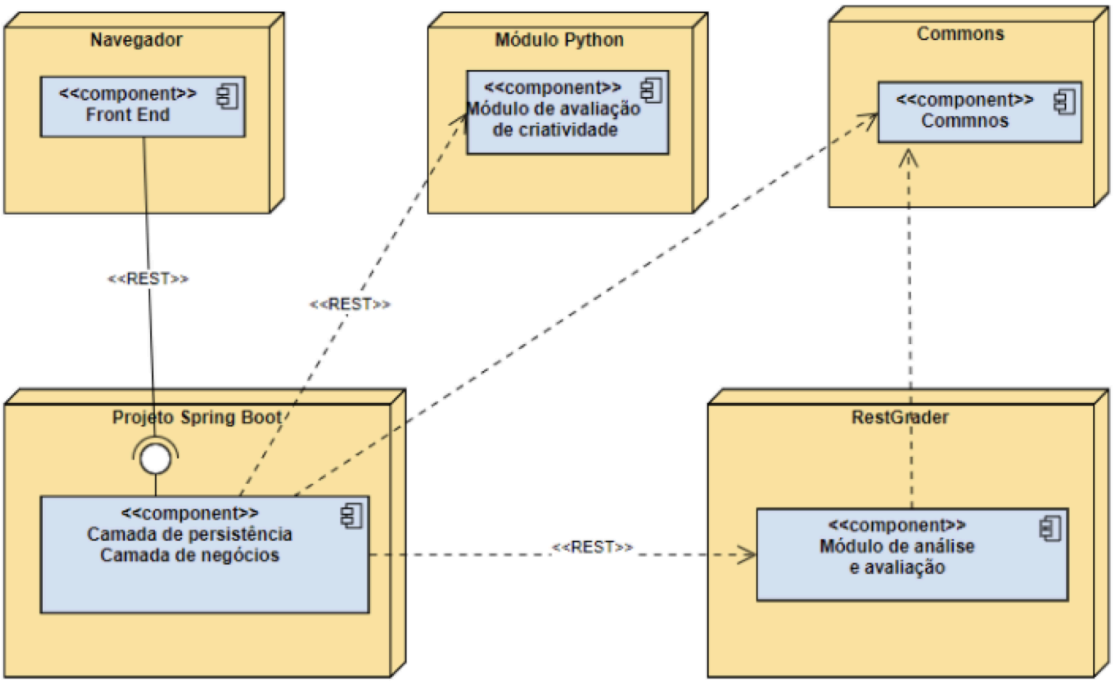
Figura 13. Interface Gráfica apresentando o resultado da avaliação do design de interface no CodeMaster.



2.4.2 Implementação do modelo de avaliação do CodeMaster - Design de interface

CodeMaster v1.0 é um sistema de software composto por cinco módulos principais nas linguagens Java (Spring), TypeScript (Angular) e Python (Schmitt, 2022).

Figura 14. Comunicação CodeMaster[8]



codemaster-angular. trata-se do *frontend* do projeto, escrito utilizando a *framework* Angular na versão 12.0.4, que chama o módulo **codemaster-apirest** através de interfaces HTTP para processar login de usuários, avaliar projetos, entre as demais funcionalidades da aplicação.

codemaster-apirest. é o *backend* do projeto escrito em Java, usando a *framework* Spring, que implementa uma API REST para tratar as requisições do módulo *frontend*. É integrado aos módulos **Appstethics** e **RestGrader** para fazer o cálculo das notas e tem conexão com um banco de dados SQL para armazenar usuários.

RestGrader. é um módulo escrito em Java integrada à API REST para computar as notas de lógica de programação e interface de usuário dos projetos enviados ao *backend*.

[8] https://codigos.ufsc.br/100000000394729/CodeMaster/-/tree/tcc/gabriel

Appsthetics. trata-se do módulo escrito em Python utilizando a biblioteca Flask que implementa uma API REST com soluções de *Machine Learning*. Esse projeto avalia a estética das telas de projetos App Inventor em formato .jpg que foram enviados ao **codemaster-apirest** e computa uma nota correspondente.

Commons. módulo implementado em Java que implementa soluções gerais utilizados pelo **codemaster-apirest** e **RestGrader**.

3. ESTADO DA ARTE

Este capítulo apresenta o estado da arte referente às abordagens que avaliam o design de interface de aplicativos móveis (criados com o App Inventor). O estado da arte é levantado por meio de um mapeamento sistemático seguindo o procedimento proposto por Petersen et al. (2015) com base no mapeamento realizado por Solecki et. al. (2020). Visa-se a responder à seguinte pergunta: Quais meios existem para a avaliação de design de interface de aplicativos móveis criados com App Inventor no contexto educacional? Observando de forma geral uma falta de soluções propostas especificamente para projetos App Inventor e/ou no contexto educacional, são consideradas também abordagens de avaliação de aplicativos móveis em geral e fora do contexto educacional.

3.1 PROTOCOLO DA REVISÃO DE LITERATURA

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

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

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

PA3. Quais são as características instrucionais do modelo/ferramenta?

PA4. Como o modelo/ferramenta foi desenvolvido(a)?

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

Com base nas perguntas de análise, foram realizadas buscas informais para a calibração da *string* de busca. Pela existência de poucos artigos relevantes ao foco da pesquisa, os termos de busca tiveram ampla abrangência para ampliar o escopo de resultados pertinentes ao estudo.

Tabela 8 – Termos de busca relevantes e sinônimos ou termos similares.

Termo	**Sinônimos ou termos similares**
“App Inventor”	Android, iOS, app, “mobile application”
Grading	Assessment, evaluation

Usability	“User Experience”, UX, “user interface”
Automated	

Dessa maneira, foi definida a *string* de busca genérica, que foi adaptada para cada base de dados conforme a sua sintaxe:

```
(“app inventor” OR app OR iOS OR Android OR “mobile application”) AND (usability OR UX OR “user interface” OR “user experience” OR design OR UI) AND (grading OR assessment OR evaluation) AND automated
```

Critérios de inclusão/exclusão. A seleção dos artigos relevantes foi feita utilizando os seguintes critérios:

- 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 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-se a abordagens para uso educacional para se ter uma visão mais ampla do estado da arte;
- Incluir artigos publicados desde a criação do iPhone (2007) até a data atual (2023);
- Não são incluídos artigos voltados a tipos específicos de apps, p.ex., apps de jogos por terem características de design de interface bem distintas de outros apps;

Fontes. As buscas foram realizadas nas principais bases de dados científicas e da área da computação: ACM Digital Library, Scopus, IEEE e Springer. Para ampliar a cobertura de publicações, além das bases de dados mencionadas, também foram conduzidas pesquisas no Google Scholar.

3.2 EXECUÇÃO DA BUSCA

As 200 publicações mais relevantes, ao termo de pesquisa, encontradas em todas as bases de dados foram analisadas e subsequentemente filtradas para atender os critérios de inclusão definidos anteriormente. O número de artigos relevantes é apresentado na Tabela 9.

Tabela 9 - Número de artigos resultantes da execução da busca

	Total de resultados da busca	**Artigos analisados**	**Artigos potencialmente relevantes**	**Artigos relevantes**
ACM	18.554	200	8	4
Scopus	561	200	9	3
IEEE	293	200	1	1
Springer	4812	200	2	1
Google Scholar	113.000	200	18	2
Total (sem duplicados)				**9**

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

No total, foram encontradas nove publicações contendo algum tipo de avaliação automatizada do design de interface de aplicativos móveis Android, iOS ou que foram desenvolvidos com App Inventor. Esses resultados estão listados na Tabela 10.

Tabela 10 – Abordagens encontradas como resultado do mapeamento.

Autoria	**Título**
Abordagens gerais	
Soui e Haddad (2023)	“Deep Learning-Based Model Using DensNet201 for Mobile User Interface Evaluation”
Dhengre et. al. (2020)	“Towards Enhanced Creativity in Interface Design through Automated Usability Evaluation”
Soui et. al. (2019)	“Assessing the quality of mobile graphical user interfaces using multi-objective optimization”

Kashif et. al. (2022)	“Development of An Automated Accessibility Evaluation Plugin Tool for Mobile Applications”
González et. al. (2012)	“BALORES: A suite of principles and metrics for graphical user interface evaluation”
Bessghaier et. al. (2021)	“On the detection of structural aesthetic defects of android mobile user interfaces with a metrics-based tool”
Liu et. al. (2022)	“Nighthawk: Fully Automated Localizing UI Display Issues via Visual Understanding”
Abordagens Educacionais	
Solecki et. al. (2019)	“CodeMasterUI Design -App Inventor: A Rubric for the Assessment of the Interface Design of Android Apps developed with App Inventor”
Solecki et. al. (2020)	“Automated Assessment of the Visual Design of Android Apps Developed with App Inventor”

Constata-se que os dois artigos de abordagem educacional dos autores Solecki *et al.* (2019) (2020) apresentam a mesma abordagem porém com métodos de avaliação diferentes.

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

De maneira geral, todas as publicações apresentam critérios de design que compõem sua respectiva avaliação. Existem muitas similaridades entre os fatores avaliados de cada abordagem. Os fatores mais comuns são regularidade, complexidade, ordenação ou sequencialidade, equilíbrio e layout.

Deve-se destacar que há abordagens com critérios de avaliação distintos dos demais, como *touchability* (Dhengre *et al*., 2020), que trata da eficiência e acurácia da interação com componentes tocáveis. Entre outros exemplos, existem critérios como repartição, ou seja, a disposição uniforme de todos os componentes nos quatro quadrantes da UI (Soui et. al., 2019).

Uma abordagem (Kashif, 2022) foca na acessibilidade, incluindo também critérios que são relevantes à usabilidade. Nessa abordagem, que implementa um plugin Figma para detectar problemas de acessibilidade, os critérios são relacionados à visibilidade dos elementos em relação ao brilho e contraste relativo.

Em todos os casos, os critérios são aplicados em uma avaliação automatizada do

design de interface de um aplicativo móvel.

Tabela 11 – Tipo dos critérios de design visual em cada abordagem.

Autoria	Fatores avaliados
Abordagens gerais	
Soui e Haddad (2023)	Estética visual, equilíbrio, layout estrutural, cores
Dhengre et. al. (2020)	regularidade (consistência e espaçamento entre componentes), complexidade (quão fácil um usuário pode encontrar a informação que busca com base no número de componentes e pontos de alinhamento na UI), *touchability* (eficiência e acurácia da interação com componentes tocáveis)
Soui et. al. (2019)	regularidade (espaçamento consistente entre componentes), composição (agrupamento visual e semântico dos componentes), ordenação (hierarquia visual, organizar componentes em ordem lógica e sequencial), complexidade (número ideal de objetos interativos), integralidade (correlação entre os componentes, referente ao tamanho, espaço entre grupos e margens), densidade (número de componentes), repartição (disposição uniforme de todos os componentes nos quatro quadrantes da UI), simetria (distribuição uniforme do número de componentes das colunas esquerda e direita da UI e componentes duplicados entre o eixo central da UI)
Kashif et. al. (2022)	Texto, contraste, brilho relativo (foco em acessibilidade)
González et. al. (2012)	Equilíbrio, linearidade, ortogonalidade, sequencialidade, regularidade
Bessghaier et. al. (2021)	Equilíbrio, economia, coesão, uniformidade, simplicidade, densidade, regularidade, sequência, homogeneidade, agrupamento, união, integralidade
Liu et. al. (2022)	Oclusão de componentes (quando um componente ou uma informação textual é ocluída), sobreposição de texto, imagem ausente, valor NULL, tela borrada
Abordagens educacionais	
Solecki et. al. (2019)	Layout, Tipografia, Escrita, Cores, Imagens
Solecki et. al. (2020)	

PA3. Quais são as características instrucionais do modelo/ferramenta?

Das únicas duas publicações que apresentam abordagens educacionais, ambas tratam da mesma ferramenta (Solecki *et al.*, 2019)(Solecki *et al.*, 2020). CodeMaster foi desenvolvido para auxiliar na avaliação da UI de aplicativos criados com App Inventor

por estudantes de educação básica. A rubrica que compõe essa abordagem foi evoluída na publicação posterior (Solecki *et al*., 2020), avaliando critérios básicos de design visual que possam ser aplicados com App Inventor. A ferramenta calcula uma nota para cada um dos critérios (Tabela 11) para que os estudantes possam tratar os problemas de design em suas interfaces.

PA4. Como o modelo/ferramenta foi desenvolvido(a)?

Dentre as abordagens das publicações, 5 de 9 envolvem o uso de *Machine Learning* como solução para avaliação automatizada de UIs de maneira geral, treinadas com conjuntos de dados de *screenshots* de aplicativos móveis. Entretanto, duas das nove abordagens (Bessghaier *et al*., 2021) (Soui *et al.*, 2019) adotaram o uso de algoritmos genéticos para encontrar problemas no design de interface, pois estes são ideais para problemas que possuem mais de uma solução. Ou seja, a detecção de defeitos visuais em interfaces móveis nessas duas abordagens foi tratada como um problema de multi-otimização. Em todos os casos, foram definidas diversas métricas para efetivamente avaliar a qualidade das interfaces de usuário, algumas sendo mais elaboradas do que outras.

Tabela 12 - Método de desenvolvimento das abordagens.

Abordagem	Método de desenvolvimento
Soui e Haddad (2023)	Um modelo de rede neural convolucional (CNN) chamado DensNet201 e um classificador *K-nearest neighbors* (KNN) foi utilizado para avaliar a usabilidade de *mobile UIs* com uma precisão de 87,4%, podendo ser usado para gerar recomendações para melhorar a usabilidade de aplicativos existentes ou orientar o design de novos aplicativos.
Dhengre et. al. (2020)	Abordagem que utiliza rede neural convolucional (CNN) para predizer três métricas de usabilidade: regularidade, complexidade e *touchability*, treinado com um *subset* do *dataset* RICO com mais de 72.000 *screenshots* de aplicativos móveis.
Soui et. al. (2019)	Diversas métricas de avaliação são incorporadas, como regularidade, composição, ordenação, complexidade, integralidade, densidade, repartição, simetria, para detectar defeitos estéticos em *mobile UIs* utilizando algoritmos evolutivos.
Kashif et. al. (2022)	Plugin do Figma desenvolvido com a proposta de resolver problemas de acessibilidade em aplicativos *mobile* automatizando *guidelines* da WCAG 2.0 e 2.1, focando especificamente em nodos de componentes no Figma.
González et. al. (2012)	Proposta de princípios estéticos e estruturais para interfaces gráficas composta por cinco métricas: equilíbrio, linearidade, ortogonalidade, sequencialidade e regularidade.

Bessghaier et. al. (2021)	Apresenta uma abordagem para detectar defeitos estéticos estruturais em interfaces de usuário de dispositivos móveis Android por meio de uma ferramenta baseada em métricas. Propõem uma análise quantitativa de diversos atributos visuais, como equilíbrio, proporção, alinhamento e simetria, para identificar possíveis problemas estéticos. A ferramenta desenvolvida é capaz de medir esses atributos automaticamente e fornecer uma pontuação de qualidade estética para cada interface analisada.
Liu et. al. (2022)	Trata-se de um modelo que detecta defeitos na interface de usuário de aplicativos através de *screenshots*. Como o treinamento de um modelo desses requer uma alta quantidade de amostragens com defeitos visuais, esse projeto também envolve a geração automática de dados de treinamento rotulados. As métricas dessa avaliação demonstram um bom desempenho quanto à detecção de defeitos visuais, com 86% dos casos detectados como resultado do experimento.
Solecki et. al. (2019)	Apresenta uma ferramenta inovadora de avaliação automatizada do design de interface de aplicativos criados com App Inventor por estudantes.
Solecki et. al. (2020)	O estudo apresenta a ferramenta CodeMaster proposta para a avaliação automatizada de UIs nos quesitos de usabilidade e estética de aplicativos criados com App Inventor no contexto educacional. Essa abordagem é baseada na definição de uma rubrica que inclui critérios de avaliação relacionados a conceitos básicos de design visual que possam ser implementados com App Inventor.

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

Deve-se destacar a importância de avaliar a qualidade dos modelos de avaliação para que sua eficácia possa ser assegurada. A grande maioria dos casos (7 de 9) utilizou técnicas estatísticas para analisar sua performance e estão apresentadas na Tabela 13. No entanto, duas das publicações não informaram o método de avaliação.

Tabela 13 – Avaliação das abordagens.

Abordagem	**Tipo de estudo**	**Fatores avaliados**	**Método de coleta de dados**	**Tamanho da amostra**	**Método de análise**	**Descobertas**
Soui e Haddad (2023)	Teste de modelo de DL	Melhores resultados de classificação	Dataset RICO com hierarquias e *screenshots* de MUIs de aplicativos Android	9677 *screenshots* de aplicativos Android do conjunto de dados RICO	Precisão, *recall*, acurácia	DensNet201 + KNN supera resultados de outras abordagens *deep learning* para o mesmo propósito.
Dhengre et. al. (2020)	Teste de modelo de DL	Performance do modelo	Dataset RICO com 72.000 *screenshots* de aplicativos *moveis*	410 imagens RGB anotadas do dataset RICO	Coeficiente de determinação (*R-squared score*)	O modelo mais bem treinado obteve *R-squared score* acima de 90% para as três métricas de usabilidade propostas.
Soui et. al. (2019)	Teste de modelo de DL/Estudo de caso	Performance dos algoritmos evolutivos testados	Questionário	24 aplicativos com 200 MUIs avaliados por 20 estudantes	Escala Likert de 7 pontos	Os resultados confirmam a eficiência do algoritmo evolutivo baseado em indicadores para detectar defeitos estéticos típicos e gerar regras de detecção.
Kashif et. al. (2022)	Adhoc	Performance das técnicas aplicadas	Revisão bibliográfica de critérios de avaliação de acessibilidade	19 artigos relevantes	*Não informado*	O plugin Figma analisa componentes de uma interface de usuário com base nos rótulos que foram fornecidos e aplica uma técnica que avalia a visibilidade dos componentes (acessibilidade).
González et. al. (2012)	*Não informado*					
Bessghaier et. al. (2021)	Estudo de caso/Teste de modelo de DL	Performance da abordagem comparada à técnicas similares	Análise estatística da performance comparativa entre abordagens similares, questionário	80 MUIs, 5 especialistas em qualidade de design gráfico	Precisão, *recall*, F1 score	Os resultados obtidos através do experimento demonstraram a eficácia da abordagem proposta, com a análise de 5 especialistas da área para validar os

						resultados da ferramenta.
Liu et. al. (2022)	Teste de DL	Performance e utilidade do modelo na detecção e localização de problemas de UI	Geração automática de bugs visuais em 64.000 imagens de 30.000 aplicativos Android	Teste do modelo realizado com 1.600 screenshots da maior plataforma de *crowdtesting* e 8.000 de screenshots gerados	Precisão, *recall*, F1-score	Nighthawk é mais performático que OwlEye e outros 13 *baselines* do estado da arte.
Solecki et. al. (2019)	Estudo de caso	Confiabilidade, validade convergente e discriminante da rubrica	Aplicativos da “App Inventor Gallery” disponíveis publicamente	978 aplicativos App Inventor	Coeficiente alpha de Cronbach, Matriz de correlação e Análise fatorial	CodeMaster demonstra ser confiável e pode ser usada para uma avaliação válida do design de interface de aplicativos App Inventor, apesar de necessitar alterações de alguns critérios da rubrica.
Solecki et. al. (2020)	Estudo de caso	Consistência interna, validade convergente e discriminante da rubrica	Aplicativos da “App Inventor Gallery” disponíveis publicamente	1.775 aplicativos App Inventor	Coeficiente alpha de Cronbach, Matriz de correlação e Análise fatorial	CodeMaster demonstra ser uma ferramenta confiável na avaliação do design visual de aplicativos App Inventor.

3.3 DISCUSSÃO

Os resultados demonstram que existem abordagens de avaliação automatizada de design de interfaces móveis como um todo, embora sejam poucas. Entretanto, fica evidente a ausência de abordagens que abrangem o contexto educacional. Nesse sentido, a grande maioria das propostas têm o propósito de auxiliar no desenvolvimento de aplicativos móveis, seja antes da etapa de desenvolvimento, como é o caso do plugin Figma para detectar problemas de acessibilidade (Kashif *et al*., 2022), ou depois, como uma ferramenta de análise de defeitos que possam ser corrigidos no ciclo de desenvolvimento do aplicativo.

De maneira geral, todas as propostas apresentam métricas sólidas de avaliação de interfaces móveis, ou seja, todas avaliam elementos do design visual desses aplicativos. Além disso, algumas publicações detalham a teoria que orienta a definição dessas métricas, inclusive com as respectivas fórmulas matemáticas.

Embora este mapeamento não tenha sido limitado apenas a propostas que avaliam aplicativos criados com App Inventor, foi encontrada somente uma abordagem educacional que foca neste tipo de projeto (Solecki *et al*., 2019)(Solecki *et al*., 2020) com base em uma rubrica de avaliação. Pela relativa simplicidade do App Inventor, por ser uma ferramenta educacional de programação em blocos, a rubrica dessas duas publicações é mais simples em comparação com as métricas encontradas nas abordagens gerais, justificada pelas limitações do ambiente de desenvolvimento em questão (App Inventor) e o estágio escolar (Educação Básica). Analisando essa abordagem, foram encontrados alguns pontos fracos como na identificação da semântica dos componentes e reconhecimento de padrões. Isso inclui critérios de avaliação como estilos tipográficos conforme a função do texto ou a consistência do estilo de ícones. Também é necessário considerar a constante evolução dos guias de estilo, especificamente o Material Design.

Contudo, este trabalho busca evoluir e complementar a rubrica do CodeMaster UI Design 1.0 conforme as atualizações do Material Design 3 e implementado métricas novas.

Ameaças a validade

O maior risco deste mapeamento sistemático trata-se da omissão de publicações relevantes a este trabalho. Como estratégia de mitigação, foi construída uma *string* de busca com todos os possíveis termos relevantes e respectivos sinônimos para aumentar o número de resultados pertinentes. Além disso, a pesquisa foi realizada em diversas bases de dados científicas para reduzir ao máximo as chances de omitir publicações relevantes.

Outra estratégia adotada para evitar riscos na seleção das publicações e na extração de dados foi a definição de critérios de inclusão e exclusão do mapeamento sistemático que foram devidamente revisados pelos dois autores e a orientadora deste trabalho.

4. EVOLUÇÃO DA AVALIAÇÃO DO UI DESIGN

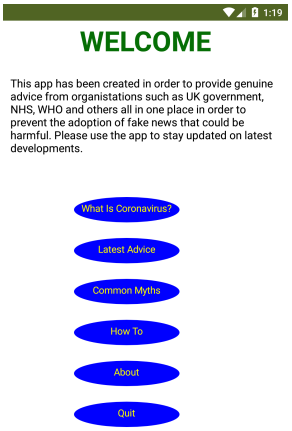
O objetivo deste trabalho é evoluir e complementar a rubrica CodeMaster UI Design – App Inventor v1.0 e a implementação e avaliação destas melhorias. Para fazer isso, em um primeiro passo são analisados erros típicos de interfaces de usuário nos apps com App Inventor.

4.1 ANÁLISE DE ERROS TÍPICOS DE DESIGN DE INTERFACE

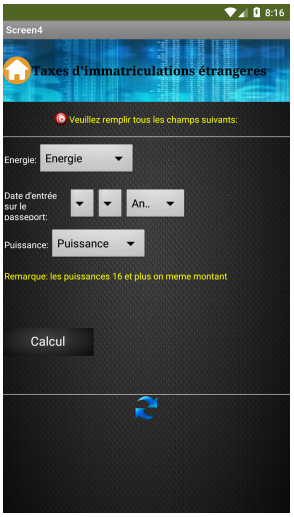
Como base para evolução da rubrica são avaliados erros típicos de design de interface de aplicativos desenvolvidos com o App Inventor. Esses erros foram observados a partir de uma análise, realizada manualmente pelos autores, de 820 *screenshots* de aplicativos disponibilizados na App Inventor Gallery e como resultados de unidades instrucionais da Computação na Escola/INCoD/INE/UFSC. Esta amostra foi selecionada por conveniência (Lima *et al.*, 2022). Exemplos de alguns *screenshots* deste tipo de app são apresentados na Figura 15.

Figura 15. Exemplos de erros típicos de design de interface

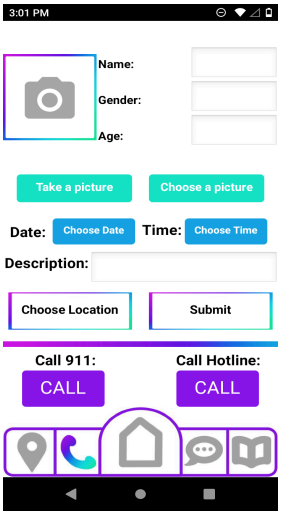
a)



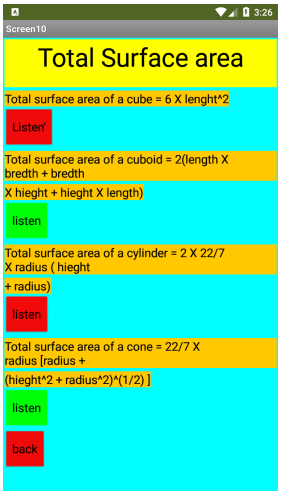
b)



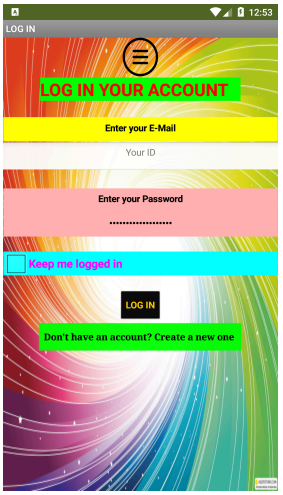
c)



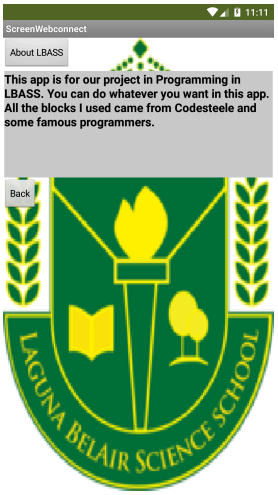
d)



e)



f)



Observa-se que estes exemplos da Figura 15, as interfaces apresentam problemas como:

a) Uso de botões no formato oval
b) Falta de alinhamento dos elementos
c) Muitos elementos em uma interface
d) Uso de cores não harmônicas
e) Uso de imagem no fundo da tela criando um efeito poluído (exceto pela tela home)
f) Uso de negrito para um parágrafo inteiro

A partir desta análise foram identificados erros típicos conforme listado na Tabela 14.

Tabela 14. Erros típicos de design de interface desenvolvidos com o App Inventor

Imagem	Imagem pixelizada
	Imagem distorcida
	Uso de imagem no fundo da tela criando um efeito poluído (exceto na tela home)
	Uso de imagens que usam paleta de cores diferente da paleta de cores do app
	Uso de imagens com *watermark* de licenças
	Uso de imagens sem fundo transparente
Ícones	Tamanho de ícones muito grandes
	Uso de ícones de diferentes familias/tipos
	Uso de ícones com cores diferentes
	Uso de ícones com tamanhos diferentes no mesmo agrupamento
Cores	Muitas cores
	Cores não agradáveis (com pelo menos duas das cores primárias RGB com valor acima de 200)
	Combinações de cores não harmônicas
	Fundo de labels não-transparente
	Falta de contraste de texto em cima de fundo
	Botões de cores diferentes
Texto	Texto muito grande
	Excesso de fontes
	Uso desnecessário de : (dois pontos) no final de labels
	Uso desnecessário de . no final de frases “finais”
	Texto (e texto em botões) começam com letra minúscula
	Alguns componentes têm ainda o texto padrão (p.ex. “Text for Button” or “hint”)
	Texto de botões muito longo
Tipografia	Uso de fonte serifada
	Uso de itálico
	Uso de negrito para um parágrafo inteiro
	Uso de fonte muito pequena/grande
	Botão com texto não centralizado

Layout	Mais de 30 elementos em uma interface (*overloaded/simplicidade*)
	Falta de alinhamento de elementos
	Inconsistência da organização de componentes e espaçamento (regularidade)
	Tela muito complexa (complexidade)
	Falta de simetria na tela (simetria)
	Falta de equilíbrio na distribuição na tela (p.ex. todos grudados no topo da tela) (balanceamento)
	Falta de consistência e uniformidade no layout de tela (uniformidade)
	Uso de botões no formato oval
	Botões com formatos diferentes
	Uso de botões com tamanhos (largura ou altura) diferentes no mesmo agrupamento (coesão)
	Uso de caixas de texto com tamanhos (largura ou altura) diferentes no mesmo agrupamento (coesão)
	Falta de espaçamento/sobreposição entre botões ou texto
	Componente próximo às bordas

4.2 REVISÃO DA RUBRICA

A avaliação de aprendizagem do aluno em relação ao design de interface de apps de App Inventor atualmente é definida por meio da rubrica CodeMaster UI Design v1.0 (Tabela 7) que foi desenvolvida por Solecki (2019) no contexto da iniciativa Computação na Escola do GQS/INCoD/INE/UFSC. A rubrica apresenta critérios para avaliação do design visual de aplicativos desenvolvidos com o App Inventor.

A rubrica foi revisada em relação aos erros típicos encontrados (Tabela 14), a versão atual do Material Design (MD3) e fatores identificados na literatura, como regularidade, complexidade, equilíbrio e layout. Na evolução da Rubrica v1.0 da ferramenta CodeMaster UI Design são mantidos alguns critérios e entram novos critérios e níveis de desempenho não abordados anteriormente. Além disso, alguns critérios e níveis de desempenho mantidos foram alterados de acordo com as orientações do Material Design 3 para continuar em conformidade com a nova versão. Como a avaliação será automatizada a partir do arquivo .aia, alguns dos erros identificados não foram expressados como critérios. O arquivo .aia não fornece alguns tipos de informações, como por exemplo informações sobre a posição dos elementos de

forma explícita. Dessa forma, alguns erros, como em relação a simetria, não são avaliados nesta versão da rubrica e outros, como alinhamento, são avaliados de forma simplificada. Essa revisão da rubrica foi feita pelos autores deste TCC em conjunto com a orientadora e pesquisadores do Computação na Escola/INCoD/INE/UFSC.

Tabela 15. Rubrica CodeMaster UI Design v2.0

Nome da métrica no CodeMaster	Problema UI	Critério	≤0 pt	1 pt	2pt
Layout					
Densidade de tela	Muitos elementos (excluindo espaço branco/alinhamentos etc.) em uma interface (*overloaded/simplicidade*)	Qual é o número mínimo e o número máximo de elementos? (L4)	min ≤ 1 ou max > 40	min > 30 e max ≤ 40	min ≥ 1 e max ≤ 30
Espaço entre componentes	Espaçamento entre elementos		Nao usou espaço em branco (label ou alinhamento vazio) entre elementos		Usou pelo menos um espaço em branco (label ou alinhamento vazio) entre elementos
Formato dos botões	Uso de botões no formato oval	Todos os botões têm forma arredondada?	Uso de botão oval	Uso de botão retangular	Uso de botão arredondado/padrão
Tamanho dos componentes alvos de toque	Componentes de “touch target” (botão, checkbox, textbox, etc.) são muito pequeno	Todos os componentes de “touch target” têm largura e altura maiores ou iguais a 10 pixels? (L1)	Não		Sim
Tamanho consistente de botões	Uso de botões com tamanhos (largura ou altura) diferentes no mesmo agrupamento (coesão/uniformidade)	Os botões agrupados na interface têm sempre o mesmo tamanho (altura e largura)? (L3)	Não possuem ou possuem em menos de 75% dos casos		Possuem o mesmo tamanho em 75% ou mais dos casos
Tipografia					
Familia da fonte	Uso de fonte serif	A família de fontes utilizada é com serifa? (T1)	Sim		Não
Uso de itálico	Uso de itálico	Tem algum texto em itálico? (T4)	Sim		Não
Uso de negrito	Uso de negrito para um parágrafo inteiro	Existe algum parágrafo de texto (mais de 3 palavras) todo em negrito?	Em menos de 50% dos casos	Entre 50% e 80% dos casos	≥ 80% dos casos
		Algum botão tem texto em negrito? MD3: Fonte do botão: Label large Roboto Medium 14	Sim		Não
Tamanho da fonte de botões	Uso de fonte muito pequeno	Todos os botões com texto têm tamanho de fonte entre 12 a 24? (T2)	Não		Sim
Tamanho da fonte de componentes		Todos os componentes com texto (exceto botões) usam tamanhos entre 10px e 96px?	Não		Sim
Botões com texto não centralizado	Botão com texto não centralizado	Existe algum botão com texto não centralizado?	Sim		Não

Quantidade de fontes	Uso de mais do que 2 fontes diferentes no app	Existem mais de 2 fontes diferentes usadas no aplicativo?	4 ou mais fontes diferentes	3	1-2 fontes
Escrita					
Capitalização de botões	Texto (e texto em botões) começam com letra minúscula	Todos os botões tem legenda que começam com letra maiúscula? MD3: Legendas de botões devem começar com letra maiúscula e as demais serem minúsculas (W1)	Em menos de 50% dos casos	Entre 50% e 80% dos casos	≥ 80% dos casos
Capitalização de sentenças		Todas as frases começam com uma letra maiúscula ou um dígito? (W2)	Em menos de 50% dos casos	Entre 50% e 80% dos casos	≥ 80% dos casos
Texto padrão dos componentes	Alguns componentes têm ainda o texto padrão (p.ex. "*Text for Button*" ou "*hint*")	Todos os componentes têm um texto diferente do padrão (por exemplo, "*Text for Button1*")? (W3)	Em menos de 80% dos casos		≥ 80% dos casos
Evitar dois pontos	Uso desnecessário de : no final das legendas	Existem legendas que terminam com ":" (dois pontos)? (W4)	Sim		Não
Evitar ponto final	Uso desnecessário de . no final de frases "finais"	Existem frases "não finais" terminando com "." (W5)	Em menos de 50% dos casos	Entre 50% e 80% dos casos	≥ 80% dos casos
Comprimento do texto de botões	Texto de botões muito longo	A maior legenda de botão tem quantos caracteres? (W6)	Mais de 24		24 ou menos
Cores					
Sistema de cores	Muitas cores	Quantas cores são usadas (além de vermelho, preto, branco e cinza)? (C1) MD3: 5 cores (tonalidade) + vermelho para erro + preto e branco	7 ou mais, ou nenhuma	6	1 a 5
Uso de cores agradáveis	Cores não agradáveis (cores neon, muito saturadas)	Existe alguma cor com saturação acima de 70% e brilho acima de 80%?	Sim		Não
Uso de cores harmônicas	Combinações de cores não harmônicas	As tonalidades de cores usadas são harmônicas entre si (monocromáticas, complementares, análogas, quadráticas, meio complementares ou triádicas)? (C4)	Não		Sim
Fundo dos labels transparentes	Fundo de labels não-transparente	Existem labels com uma cor de fundo não transparente?	Sim		Não
Contraste entre texto e cor de fundo	Falta de contraste de texto em cima de fundo	Qual é o nível WCAG do aplicativo em relação ao contraste do texto? (C2) MD3: Texto grande (18pts ou maior, ou 14pts ou maior se estiver em negrito) deve atender a uma taxa de contraste de 3:1 para acessibilidade padrão (classificada como AA) e 4.5:1 para maior acessibilidade (classificada como AAA); Os tamanhos padrão do corpo do texto (menores que 18pts ou menores que 14pts se estiver em negrito) devem atender a uma taxa de contraste de 4.5:1 para acessibilidade padrão (classificada como AA) e 7:1 para maior acessibilidade (classificada como AAA). (espaços de cores HSL, HCL ou HCT comparando luminosidade ou tom)	Menos de 80% dos componentes são pelo menos Nível AAA	Mais de 80% dos componentes são pelo menos Nível AAA	Todos os componentes são pelo menos Nível AAA

Botões com mesma cor de fundo	Botões de cores diferentes	Existem botões preenchidos com cores diferentes?	Mais de 4 cores		Até 4 cores
		Imagens e ícones			
Pixelização	Imagens pixeladas	Existem imagens pixeladas?	Sim		Não

5. IMPLEMENTAÇÃO E TESTE DO CODEMASTER - UI DESIGN 2.0

Este capítulo apresenta a implementação e teste da ferramenta CodeMaster UI Design 2.0.

5.1 ANÁLISE DOS REQUISITOS

Foram revisados e atualizados os requisitos funcionais originalmente especificados por Solecki (2020). A princípio são mantidos todos os requisitos do CodeMaster UI Design v1.0, modificando somente os critérios de avaliação e os níveis de desempenho conforme a nova versão da rubrica.

Tabela 18. Requisitos funcionais para o CodeMaster UI Design 2.0

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 15 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 15. 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 15.	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

Foram também identificados os requisitos não funcionais conforme apresentado na tabela 19, analisando a versão 1.0 do Codemaster UI Design como também a atualização da infraestrutura do CodeMaster em geral (Schmitt, 2022).

Tabela 19. Requisitos não funcionais para o CodeMaster UI Design 2.0

ID	**Requisito**	**Descrição**
RNF1	Linguagem de programação Java para o back-end	O back-end da ferramenta deve ser desenvolvido na linguagem de programação Java, seguindo a estrutura existente do CodeMaster. Essa ainda é uma linguagem bem conhecida, o que facilita a manutenção do sistema
RNF2	Linguagem de programação Typescript com Angular para front-end	O front-end da ferramenta deve ser desenvolvido na linguagem de programação Typescript, usando a estrutura Angular e mantendo a estrutura existente do CodeMaster, uma combinação amplamente usada que facilita e acelera o desenvolvimento na Web
RNF3	Linguagem de programação Python para avaliação	A avaliação deve ser implementada em python, mantendo a estrutura do CodeMaster existente atualmente, pois facilita o desenvolvimento envolvendo modelos de Machine Learning
RNF4	Comunicação REST	As partes do sistema (front, back e avaliações) devem se comunicar por meio de APIs REST
RNF5	Criptografia não reversível do e-mail do aluno	Uso de email como chave e identificador para manter a pontuação do aluno. Seguindo as práticas recomendadas, e para manter a privacidade e a segurança dos dados, o e-mail deve passar por uma criptografia não reversível (hash) para ser salvo no banco de dados.
RNF6	Identidade Visual CnE	A interface do usuário deve seguir os padrões atuais de identidade visual do CodeMaster, conforme definido pela iniciativa Computação na Escola
RNF7	Sistema Web	A ferramenta deve ser acessada por meio de um navegador da Web de PC, com conexão à Internet.

5.2 MODELAGEM DA ARQUITETURA DO SISTEMA

As melhorias sugeridas para a ferramenta CodeMaster foram desenvolvidas mantendo o modelo arquitetônico da versão existente descrita na seção 2.4.2. Para a evolução do CodeMaster v1.0, utiliza-se a linguagem de programação Java no backend integrada à *framework* Spring para atuar nos módulos **RESTGrader**, **codemaster-apirest** e **Commons**. Essas linguagens e tecnologias foram utilizadas por já terem sido utilizadas no desenvolvimento da versão anterior.

Para a implementação da rubrica atualizada do CodeMaster UI Design – App Inventor, foram adicionadas diversas classes referentes às métricas da rubrica no diretório **appinventorgrader.conceitos** do módulo **RESTGrader**. Essas novas métricas em seguida foram integradas à API para que sejam refletidas no *frontend*. Além disso, algumas métricas existentes foram alteradas ou excluídas tanto para atender às especificações atualizadas da rubrica e mitigar dificuldades de avaliar determinados aspectos de um projeto por conta das limitações técnicas do MIT App Inventor.

Por último, no dockerfile do *backend,* foi implementada uma cache de dependências para acelerar o tempo de build desses respectivos módulos, que teve uma melhoria de até 80% (de 15-20 minutos para 3-5 minutos). Essa alteração foi crucial para melhorar a experiência de desenvolvimento visto que esses pacotes (dependências) eram reinstalados desnecessariamente a cada build.

5.3 MODELAGEM DETALHADA E IMPLEMENTAÇÃO

Para a implementação da rubrica CodeMaster UI Design – App Inventor, são realizadas as seguintes alterações no módulo **RESTGrader** para implementar a nova rubrica:

- São criadas as métricas de **botões com texto não centralizado, cores dos botões, fundo dos labels, quantidade de fontes, cores agradáveis,** e **espaço entre componentes** como detalhados na tabela 15**.**
- São ajustadas as métricas de **capitalização de botões, capitalização de sentenças, formato dos botões, sistema de cores, uso de negrito, texto padrão dos componentes, tamanho consistente dos botões, evitar ponto final** e **comprimento do texto de botões.** No caso das métricas referentes à capitalização, tamanho dos botões, uso de negrito, texto padrão, ponto final e texto de botões tiveram suas avaliações flexibilizadas, permitindo uma tolerância de até 80% de conformidade (especificados na Tabela 15) para não prejudicar as notas de aplicativos de referência.
- São removidas as métricas de **cores do material design, ícones do material design** e **distorção de imagens.** As métricas referentes às cores e ícones

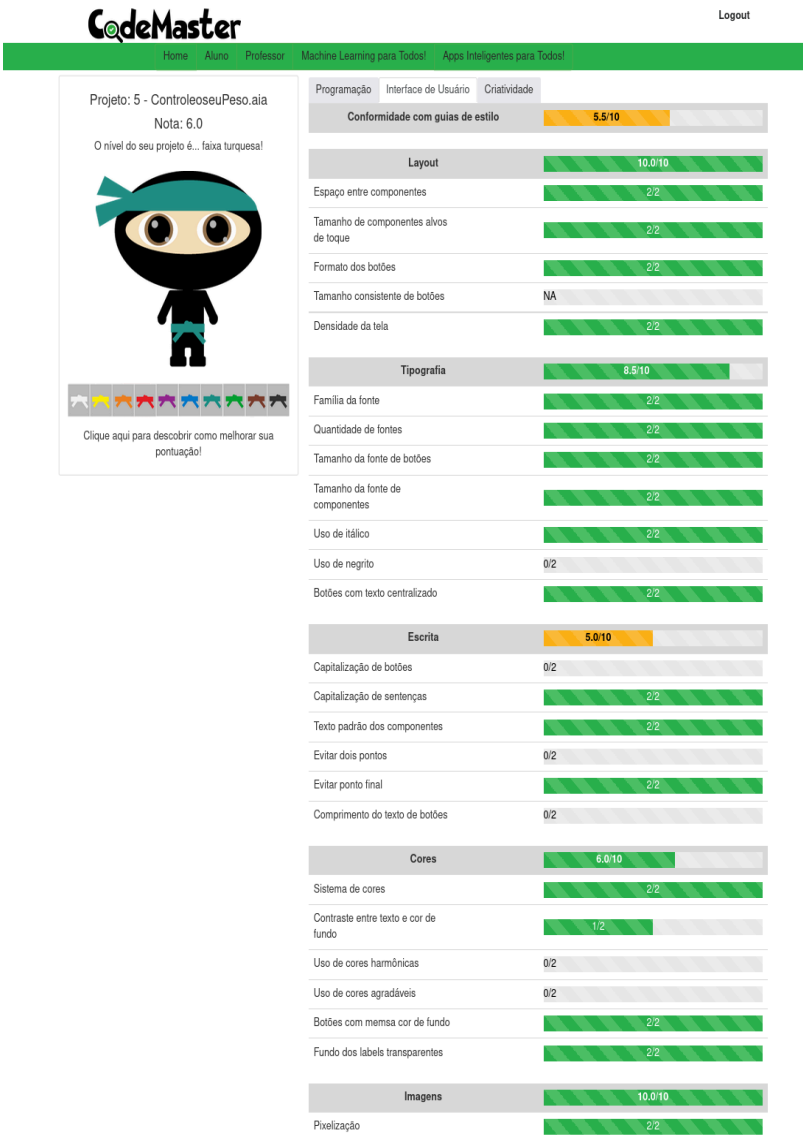
(implementadas no contexto do Material Design 2) foram removidas pois não são mais aplicáveis no contexto do Material Design 3. Também foi removida a métrica referente à distorção de imagens, composto pelas três métricas: distorção de imagens em botões, do componente imagem e de imagens de fundo da tela. Essa remoção foi necessária pois a avaliação desse critério recebeu nota zero em ambos os apps de referência (XôDengue e Yu). Desde o início do projeto, foi definido que não seriam alteradas ou adicionadas novas métricas referentes da secção de imagens pela complexidade da implementação e/ou manutenção desses critérios.

- Foi aplicado peso 4 em métricas específicas, cujos resultados são prioritários para determinar a conformidade da UI, estas sendo **componentes próximos a borda, uso de negrito, espaço entre componentes, uso de itálico, capitalização de botões, evitar dois pontos, contraste entre texto e cor de fundo, sistema de cores, fundo dos labels, cores harmônicas** e **cores agradáveis**.

- O cálculo das médias das categorias e da média final da interface foi ajustada para não contabilizar métricas avaliadas como N/A.

- Foi aplicado o modelo de arredondamento da UFSC na média final da interface cujo valor é arredondado em intervalos de 0,5.

Figura 16. Exemplo de avaliação de UI com a rubrica 4.0



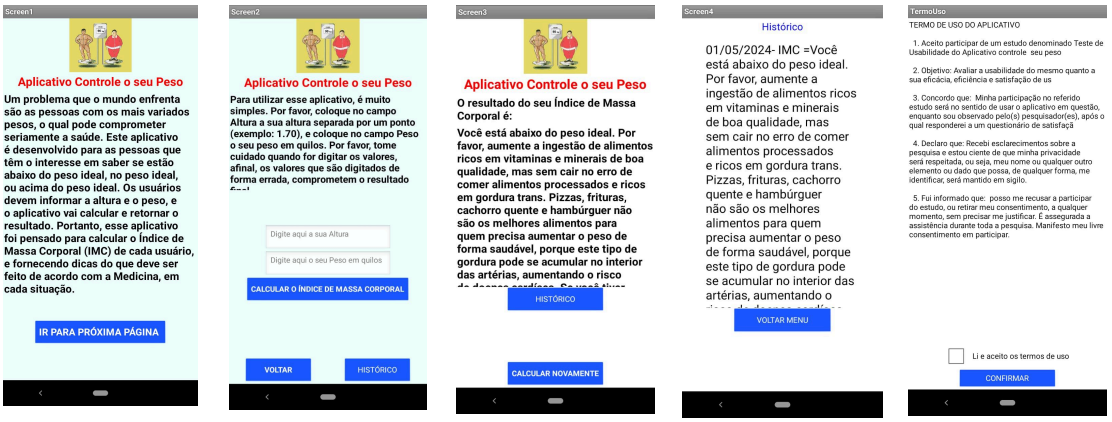
6. AVALIAÇÃO DA CONCORDÂNCIA DO MODELO DE AVALIAÇÃO

Com o objetivo de avaliar a qualidade da nova versão da rubrica proposta foi realizada uma avaliação da concordância da nota gerada automaticamente pelo modelo de avaliação implementado em comparação com uma nota alocada por um modelo de avaliação da estética visual desenvolvido por Lima (2022). O modelo de avaliação da estética visual é um modelo de *deep learning* projetado para quantificar a estética visual de interfaces de usuário móveis Android. O modelo utiliza uma rede neural convolucional (CNN) treinada com *screenshots* de interfaces de aplicativos Android, empregando uma abordagem de aprendizado supervisionado baseado em regressão. Testes demonstraram um desempenho adequado, apresentando um erro quadrático

médio de 0,022 ao classificar as imagens. Esse resultado indica que as previsões do sistema estão próximas dos valores reais, com uma precisão significativa.

Para esta avaliação foram selecionados 5 aplicativos criados com App Inventor no contexto da iniciativa Computação na Escola/INCoD/INE/UFSC e/ou disponibilizados na App Inventor Gallery, com designs de interface de usuários bons e ruins.

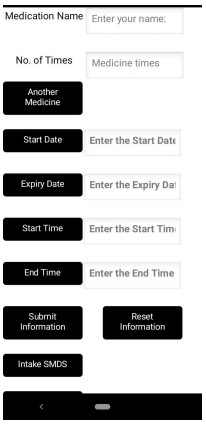
Tabela 20. Apresentação dos aplicativos testados

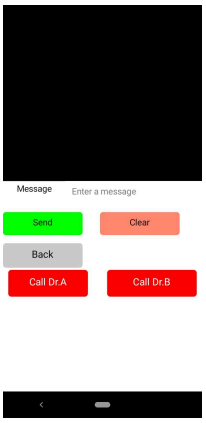
Controle o seu peso

Controleoseupeso

Yu

Yu

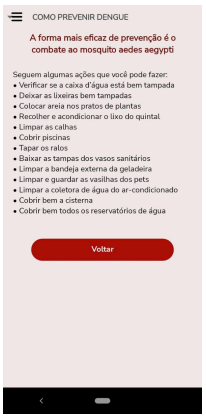
Smart Medicine

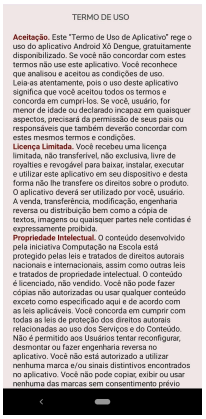


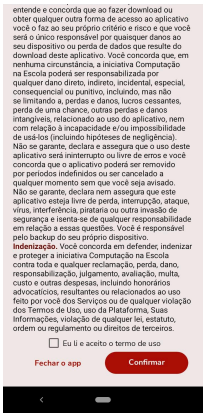


Smart Medicine

Xô dengue







Xo dengue

Covidhope

Covidhope

Cada um dos aplicativos foi avaliado pelo sistema implementado do modelo evoluído Codemaster - UI Design v2.0 e pelo modelo de avaliação da estética visual (Lima, 2023). As notas resultantes sao apresentados na Tabela 21.

Tabela 21. Notas do CodeMaster 2.0 e do modelo de avaliação de estética visual

	Controle o seu peso	**Covidhope**	**Smart Medicine**	**Xô Dengue**	**Yu**
Nota CodeMaster UI Design 2.0	5,5	4	4,5	9,5	10
Avaliação Estética (Lima, 2022)	4,6	4,25	5,4	6,14	7,6

Com base nestes dados, foi analisado o *interrater agreement* (IRA) e a *interrater reliability* (IRR). *Interrater agreement* refere-se ao grau em que o CodeMaster UI

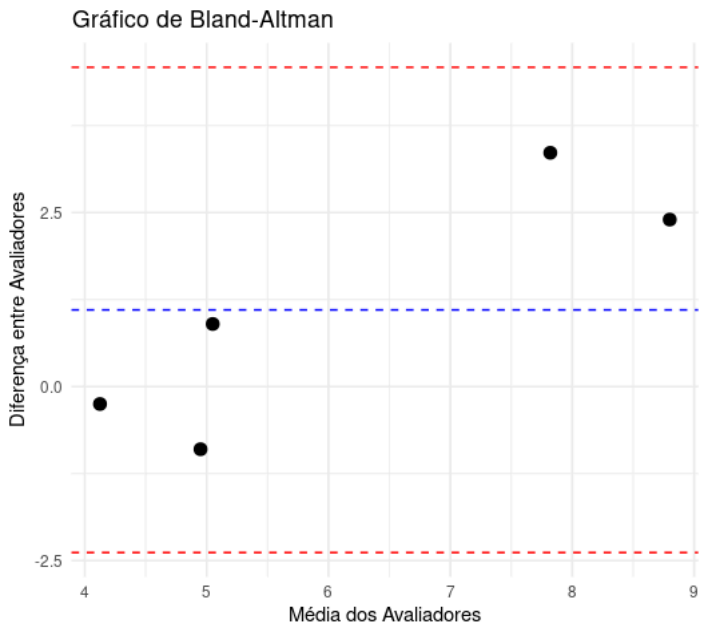
Design 2.0 e a Avaliação Estética atribuem os mesmos valores exatos para cada item sendo avaliado. Isso significa que, se todos os avaliadores estiverem em completo acordo, as avaliações individuais para um item específico seriam idênticas. Em contraste, *interrater reliability* refere-se ao grau em que os avaliadores conseguem distinguir consistentemente entre diferentes itens em uma escala de medição. Isso implica que, mesmo que os avaliadores não atribuam exatamente os mesmos valores para um item específico, eles devem ser capazes de manter uma consistência relativa na diferenciação entre itens ao longo de várias avaliações.

Para avaliar a concordância entre as notas geradas pelo CodeMaster UI Design 2.0 e o modelo de avaliação de Lima (2022), foram utilizados os seguintes métodos estatísticos: Intraclass Correlation Coefficient (modelo ICC3) e o gráfico de Bland-Altman.

O Intraclass Correlation Coefficient (ICC) é uma medida estatística utilizada para avaliar a consistência ou concordância entre diferentes avaliadores. O modelo ICC3 é apropriado para esta análise porque todos os 5 aplicativos foram avaliados por ambos os avaliadores (CodeMaster e Modelo de estética visual). A análise do coeficiente de correlação intraclasse (ICC3) resultou em um valor de 0,6791. De acordo com a classificação de Cichetti (1994), correlações entre 0,75 e 1,0 são consideradas excelentes; de 0,6 a 0,75 são boas; de 0,4 a 0,6 são razoáveis; e menores que 0,4 são consideradas pobres. Portanto, o valor obtido de 0,6791 indica uma correlação boa entre as avaliações.

O gráfico de Bland-Altman é uma abordagem gráfica utilizada para avaliar a concordância entre duas medidas. Este método envolve a plotagem da diferença entre as duas medidas contra a média das duas medidas. A análise de Bland-Altman revelou uma diferença média de 1,102, indicando um viés em que, em média, o Avaliador 1 atribui notas 1,102 unidades mais altas que o Avaliador 2. Os limites de concordância foram de -2,3849 (inferior) a 4,5889 (superior). Essa ampla variabilidade sugere que os critérios de avaliação podem não estar suficientemente padronizados.

Figura 17. Gráfico de Bland-Altman



Os resultados mostram que há algum nível de concordância entre as notas dadas pelo Codemaster e pelos Avaliadores de Estética, mas ainda há espaço para melhorias na consistência das avaliações. Isso sugere que o Codemaster ainda precisa de ajustes para alinhar suas avaliações mais de perto com as percepções humanas de estética.

6.2 DISCUSSÃO DOS RESULTADOS

Estes resultados de forma geral fornecem uma primeira indicação que o Codemaster UI Design v2.0 atualmente já consegue com uma correlação boa avaliar a qualidade do design de interfaces de apps App Inventor, mas ainda existe possibilidade de ajustes para alinhar suas avaliações mais de perto com as percepções humanas de estética de acordo com o modelo de estética visual.

As divergências entre as notas absolutas atribuidas pelos avaliadores podem ser justificadas por alguns fatores. Devido a limitações intrínsecas ao App Inventor, não foi possível avaliar alguns critérios importantes que influenciam significativamente a estética do aplicativo, como o alinhamento de todos os componentes, simetria, regularidade e o espaçamento (*padding*) da tela. Outra justificativa é que a avaliação estética e o CodeMaster não avaliam a necessariamente a mesma coisa, sendo a avaliação estética bem mais ampla.

O aplicativo "Controle o seu peso" recebeu uma nota total de 5,5, um valor superior ao esperado em comparação com as notas de estética. Esta discrepância pode ser justificada pelo fato de que o aplicativo atende a um grande número de critérios estabelecidos, resultando em uma nota mais elevada, mesmo com uma estética visual com pouca atratividade. Na média dos critérios de layout, o aplicativo obteve a nota máxima de 10. Contudo, a estética do aplicativo, avaliada de forma qualitativa, é prejudicada pela falta de espaçamento (*padding*) e algumas falhas na regularidade da tela.

Este problema na avaliação do layout também é evidente no aplicativo "Smart Medicine", que apresenta várias falhas em termos de alinhamento, regularidade, simetria e balanceamento, mas ainda assim recebeu uma nota de 6 no critério de layout.

O aplicativo “Yu” recebeu a nota 10 na avaliação do CodeMaster e nota 7 na avaliação de estética visual (a nota mais alta das avaliações de estética realizadas). Isto indica que o design deste aplicativo, mesmo estando em conformidade com as diretrizes avaliadas, está apresentando somente uma boa atratividade visual, o que pode ser pela falta da avaliação de alguns elementos de imagens ou ícones. Como uma oportunidade de melhoria, sugere-se a inclusão de novos critérios de layout que atualmente não são avaliados, devido à falta de informações suficientes fornecidas pelo App Inventor para essas avaliações em análises estáticas de código. A implementação de um modelo de deep learning para avaliar esses critérios poderia aumentar a precisão da avaliação.

Outra área de melhoria refere-se à avaliação de imagens e ícones. Atualmente, é possível avaliar a qualidade das imagens em termos de distorção. No entanto, não é possível verificar fatores de qualidade do ícone adicionado ao aplicativo, como uso de marcas d'água, nem a conformidade das cores com o restante do aplicativo. A inclusão de mecanismos para avaliar esses aspectos pode aprimorar a qualidade da avaliação visual dos aplicativos.

7. CONCLUSÃO

Como resultado deste trabalho foram analisados detalhadamente, a fundamentação teórica sobre o ensino e a aprendizagem do design de interface de usuário na educação básica, o desenvolvimento de design de interface com App Inventor, os guias de estilo de design de interface de apps Android e o CodeMaster UI Design. Foi também conduzida uma análise do estado da arte sobre modelos de avaliação automatizada de design de interface de apps criados com App Inventor no contexto da educação básica, identificando oportunidades de aprimoramento do projeto.

Com base nessas análises, o modelo de avaliação CodeMaster - Design de UI foi sistematicamente revisado e evoluído, resultando em uma rubrica com atualizações dos guias de estilo mais recentes. Os ajustes implementados no modelo de avaliação foram testados e os resultados indicaram já uma boa correlação entre as notas alocadas pela implementação da nova versão do modelo e um modelo de avaliação de estética visual (Lima, 2022).

Esta evolução do modelo de avaliação de abordagem automatizada pode fornecer aos alunos uma avaliação objetiva, minimizar o esforço dos professores e auxiliar aqueles sem formação específica em computação. Assim, 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 aprimorada tornando a avaliação de layout mais precisa com a adição de critérios de alinhamento, regularidade, simetria e balanceamento. Outra melhoria pode ser na avaliação de imagens, incorporando a capacidade de reconhecer características dos ícones, como *watermark* de licença e cores.

Referências

AHN, S. **A Study on Information Science Curriculum of Productivity Tools to Increase Ability for Problem Solving in Elementary and Middle School.** Journal of The Korean Association of Information Education, 18(2), 235-242, 2014.

AIGA. **Graphic design curriculum.** 2022. Disponível em: https://www.aiga.org/resources/academic-design-education/graphic-design-curriculum. Acesso em: setembro de 2022.

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.; 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.

ALVES, N. da C., GRESSE VON WANGENHEIM, C., HAUCK, J. C. R., BORGATTO, A. F. **A Large-scale Evaluation of a Rubric for the Automatic Assessment of Algorithms and Programming Concepts.** Proc. of the 51st ACM Technical Symposium on Computer Science Education, Portland/USA, p. 556–562, 2020.

ALVES, N. da C., GRESSE VON WANGENHEIM, C., HAUCK, J. C. R. **Assessing Mobile Apps Creativity in Computing Education.** Tese (Doutorado em Ciência da Computação) – PPGCC/Universidade Federal de Santa Catarina, 2023.

ATMOS. **LCH is the best color space!.** 2023. Disponível em: https://atmos.style/blog/lch-color-space. Acesso em: junho de 2023.

BESSGHAIER, N.; SOUI, M. *et al.* **On the detection of structural aesthetic defects of android mobile user interfaces with a metrics-based tool.** ACM Transactions on Interactive Intelligent Systems, vol. 11, no. 3, 2021.

BIAGIOTTI, L. C. M. **Conhecendo e Aplicando Rubricas Em Avaliações.** In: Proc. of the 12º Congresso Internacional de Educação a Distância, Florianópolis, Brasil, 2005.

BORDINI, A. *et al.* **Computação na educação básica no brasil: o estado da arte.** Revista de Informática Teórica e Aplicada, 23 (2), 210-238, 2016.

COLOR MATTERS. **Basic Color Theory.** 2019a. Disponível em: https://www.colormatters.com/color-and-design/basic-color-theory. Acesso em: abril 2023.

COLOR MATTERS. **Color Systems - RGB & CMYK.** 2019b. Disponível em: https://www.colormatters.com/color-and-design/color-systems-rgb-and-cmyk. Acesso em: abril 2023.

CSTA. **K–12 Computer Science Framework.** 2016. Disponível em: http://www.k12cs.org. Acesso em: setembro de 2022.

DE LUCA, C.; KLINGER, D.A. A**ssessment literacy development: identifying gaps in teacher candidates' learning.** Assessment in Education: Principles, Policy & Practice. Vol. 17, 2010.

DEV. Color harmonies in javascript. 2019. Disponível em: https://dev.to/benjaminadk/make-color-math-great-again--45of Acesso em: junho de 2023.

DHENGRE, S.; MATHUR, J. et. al. **Towards Enhanced Creativity in Interface Design through Automated Usability Evaluation.** International Conference on Computational Creativity. Coimbra, Portugal, 2020.

FALKEMBACH, G. A. M.; AMORETTI, M. S. M.; TAROUCO, L. R.; VIERO, F. **“Aprendizagem de Algoritmos: Uso da Estratégia Ascendente de Resolução de Problemas”.** 8º Taller Internacional de Software Educativo. Santiago: Chile, 2003.

FERREIRA, M. N. F.; GRESSE VON WANGENHEIM,C.; MISSFELDT FILHO, R.; DA CRUZ PINHEIRO, F.; HAUCK, J. C. R. **Learning user interface design and the**

development of mobile applications in middle school. ACM Interactions, 26 (4), 2019.

GOMES, T.; MELO, J. **App Inventor: Android para Não Programadores.** II Simpósio Brasileiro de Tecnologia da Informação. Recife, 2013.

GONZÁLEZ, S.; MONTERO, F. et. al. **BALORES: A suite of principles and metrics for graphical user interface evaluation.** Automated Assessment of the Visual Design of Android Apps Developed with App Inventor. Conferência Internacional sobre Interaccion Persona-Ordenador, Elche, Espanha, 2012.

GOOGLE. **Material design.** 2023. Disponível em https://m3.material.io/. Acesso em: março de 2023.

GRESSE VON WANGENHEIM, C.; HAUCK, J. C. R.; DEMETRIO, M. F.; PELLE, R. ALVES, N. d. C.; BARBOSA, H.; AZEVEDO, L. F. **CodeMaster – Automatic Assessment and Grading of App Inventor and Snap! Programs.** Informatics in Education, vol. 17, no. 1, 2018, 117-150.

HATTIE, J.; H, TIMPERLEY. **The power of feedback. Review of Educational Research.** Vol. 77, 2007

KASHIF, S.; AHMAD, M. et. al. **Development of An Automated Accessibility Evaluation Plugin Tool for Mobile Applications.** Proc. of the 3rd International Conference on Innovations in Computer Science & Software Engineering, Casablanca, Marrocos, 2022.

KELLEHER, C.; PAUSCH, R. **Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers.** ACM Computing Surveys (CSUR), 37(2):83–137. Publisher: ACM New York, NY, USA, 2005.

LEE, I.; MARTIN, F.; DENNER, J.; COULTER, B.; ALLAN, W.; ERICKSON, J.; MALYN-SMITH, J.; WERNER, L. **Computational thinking for youth in practice. ACM Inroads**. Vol. 2, 2011.

LIMA, A.; MARTINS, O. P. H. R..; GRESSE VON WANGENHEIM, C.; VON WANGENHEIM, A.; BORGATTO, A. F.; HAUCK, J. C. R. **Automated assessment of visual aesthetics of Android user interfaces with deep learning.** Sociedade Brasileira de Computação, 2022.

LIMA, A. L. S.; GRESSE VON WANGENHEIM, C. **Visual aesthetics of App Inventor app screenshots dataset v2.0.** GQS/INCoD/INE/UFSC, 2022. Disponível em: https://gqs.ufsc.br/visual-aesthetics-of-app-inventor-app-screenshots-dataset-v2-0/. Acesso em: 10 jul. 2023.

LIU, Z.; CHEN, C. et. al. **Nighthawk: Fully Automated Localizing UI Display Issues via Visual Understanding.** IEEE Transactions on Software Engineering, vol. 49, no. 1, 2022.

KELLEHER, C.; PAUSCH, R. **Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers.** ACM Computing Surveys, v. 37, n.2, p. 83–137, 2005.

MEC. **Base Nacional Comum Curricular.** 2023. Disponível em: http://basenacionalcomum.mec.gov.br. Acesso em: junho de 2023.

MEC. **Normas sobre Computação na Educação Básica – Complemento à Base Nacional Comum Curricular (BNCC).** Parecer 02/2022 CNE/CEB/MEC. 2022. Disponível em: http://portal.mec.gov.br/component/content/article/323-secretarias-%20112877938/orgaos-vinculados-82187207/12992-diretrizes-para-a-educacao-basica?Itemid=164. Acesso em: junho de 2023.

MIT - Massachusetts Institute of Technology. **“App Inventor for Android”.** 2023. Disponível em: appinventor.mit.edu. Acesso em: agosto de 2022.

DE OLIVEIRA, M; DE SOUZA, A; FERREIRA, A; BARREIROS, E. **Ensino de lógica de programação no ensino fundamental utilizando o Scratch: um relato de experiência.** Workshop sobre educação em computação. Sociedade Brasileira de Computação, p. 239-248, 2014.

PATTON, EW; TISSENBAUM, M; HARUNANI, F. **MIT App Inventor: Objectives, Design, and Development.** Computational Thinking Education. Springer, Singapore, 2019.

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.** Anais do Computer on the Beach. Florianópolis, Brasil, 2018.

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.

SBC. **Diretrizes para ensino de Computação na Educação Básica. Sociedade Brasileira de Computação.** 2018. Disponível em: http://www.sbc.org.br/educacao/diretrizes-para-ensino-de-computacaona-educacao-basica. Acesso em: outubro de 2022.

SCHMITT, T. **Refatoração da Ferramenta CodeMaster.** Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade Federal de Santa Catarina, 2022.

SOLECKI, I. S. S. **Uma abordagem para a avaliação do design visual de aplicativos móveis criados com linguagens de programação baseadas em blocos.** Programa de Pós-Graduação em Ciência da Computação. Universidade Federal de Santa Catarina. Florianópolis, 2020.

SOLECKI, I.; PORTO, J. A.; ALVES, N. d. C., GRESSE VON WANGENHEIM, C., HAUCK, J. C. R., BORGATTO, A. F. **Automated Assessment of the Visual Design of**

Android Apps Developed with App Inventor. Proc. of the 51st ACM Technical Symposium on Computer Science Education, Portland, USA, 2020.

SOLECKI, I.; PORTO, J. et. al. CodeMasterUI Design -App Inventor: A Rubric for the Assessment of the Interface Design of Android Apps developed with App Inventor. XVIII Brazilian Symposium on Human Factors in Computing Systems, Vitória, Brazil, 2018.

SOUI, M.; CHOUCHANE, M. et. al. **Assessing the quality of mobile graphical user interfaces using multi-objective optimization.** Soft Computing, vol. 24, 2020, 7685–7714.

SOUI, M.; HADDAD, Z. **Deep Learning-Based Model Using DensNet201 for Mobile User Interface Evaluation.** International Journal of Human–Computer Interaction, vol. 39, no. 9, 2023.

TIC EDUCAÇÃO. **Marco Referencial Metodológico para a Medição do Acesso e Uso das Tecnologias de Informação e Comunicação (TIC) na Educação.** Centro Regional de Estudos para o Desenvolvimento da Sociedade da Informação, 2016.

TUNIN, A. S. M; HENRIQUE, M. P; BAIRRA, M. A. **Políticas de difusão das tecnologias da informação e comunicação na educação: reflexões a partir de um resgate histórico.** Revista ensaios e pesquisas em educação e cultura, vol. 3, núm. 4, 2022.

VALENTIM, N. M. C.; SILVA, W. A. F.; CONTE, T. **Avaliando a Experiência do Usuário e a Usabilidade de um Aplicativo Web Móvel: um relato de experiência.** XVIII Congresso Ibero-Americano em Engenharia de Software, Lima, Peru, 2015.

WENDELL, B. G. **Programar é bom para as crianças? Uma visão crítica sobre o ensino de programação nas escolas.** Texto Livre: Linguagem e Tecnologia, vol. 7, núm. 2, 2014.

WEST-KNIGHTS, I. **Why are schools in China looking west for lessons in creativity?** Financial Times, 2017.

XING, B.; SI, H.; CHEN, J.; YE, M.; SHI, L. **Computational model for**

predicting user aesthetic preference for GUI using DCNNs. CCF Transactions on Pervasive Computing and Interaction, vol. 3, no. 2, pp. 147–169, 2021.

Apêndice A

Repositório do CodeMaster: **https://codigos.ufsc.br/100000000394729/CodeMaster**

Apêndice B

Evolução de um Modelo de Avaliação de Design de Interface no Contexto do Ensino da Computação com o App Inventor

Martina Klippel Brehm[1], Gabriel Andrade Borges Nascimento[1], C. Gresse von Wangenheim[1], Jean C. R. Hauck[1], Adriano L. S. Lima[1]

[1]Departamento de Informatica e Estatística, Universidade Federal de Santa Catarina (UFSC), Florianópolis, Brasil

{gabriel.b.n,martina.brehm}@grad.ufsc.br,
{c.wangenheim,jean.hauck,adriano.borgatto}@ufsc.br

Abstract. *Information technology has greatly expanded in recent decades, making the inclusion of basic computing principles in basic education important. One way to teach computing is by developing applications for Android devices using App Inventor, which also addresses interface design. To simplify the evaluation process, this study evolves the CodeMaster-UI Design tool, which automates the evaluation of student app projects using a rubric. The results indicated a good correlation (ICC3 = 0.6791) between the evaluations, although there is room for improvement.*

Resumo. *A tecnologia da informação teve uma grande expansão nas últimas décadas e, por causa disso, a inclusão de princípios básicos da computação na educação básica torna-se importante. Uma maneira de ensinar computação é pelo desenvolvimento de aplicativos para dispositivos Android utilizando App Inventor que aborda também o design da interface. Para simplificar o processo de avaliações, este estudo evolui a ferramenta CodeMaster-UI Design, que automatiza a avaliação de projetos de aplicativos estudantis utilizando uma rubrica. Os resultados indicaram uma correlação boa (ICC3 = 0,6791) entre as avaliações, embora haja espaço para melhorias.*

1. Introdução

Sistemas de software atualmente são parte fundamental de muitas atividades do dia-a-dia (Salim *et al*., 2015) e o número de pessoas que estão utilizando as aplicações móveis tem aumentado consideravelmente (Valentim *et al*., 2015). Deste modo, é fundamental que as pessoas possam ter conhecimentos básicos de computação desde o início da vida escolar (Oliveira *et al*., 2014).

Uma das iniciativas que buscam incorporar as tecnologias da informação e comunicação na educação brasileira é a Computação na Escola/INCoD/INE/UFSC[9] que visa o ensino de design de interface junto com a programação na educação básica (Ferreira *et al*., 2019).

Uma maneira de ensinar computação é pelo desenvolvimento de aplicativos para dispositivos Android utilizando App Inventor. Considerando que a usabilidade e o design de interface estão entre os importantes fatores de qualidade de aplicativos

[9] www.computacaonaescola.ufsc.br

móveis, o ensino de computação por meio do desenvolvimento de apps com App Inventor, permite também ensinar conceitos de design de interface (Ferreira *et al.*, 2019).

Para facilitar a avaliação no ensino de computação na educação básica é interessante automatizar a avaliação (Alves *et al*., 2019). A avaliação da aprendizagem dos alunos na área de computação e design de interface pode ser uma tarefa complexa e trabalhosa para os professores por diversos fatores. Um dos fatores é que muitos professores que ensinam informática não possuem formação para tal (De Luca et al., 2010) podendo ter dificuldades em realizar uma avaliação manual da forma correta. Outro benefício da avaliação automatizada é que ela proporciona imparcialidade.

Com esse objetivo, já foi criada a ferramenta web CodeMaster UI Design v1.0 (Solecki, 2020), que permite automaticamente avaliar o design de interface de aplicativos desenvolvidos com o App Inventor como resultado da aprendizagem, com base em uma rubrica. A ferramenta atualmente apresenta a pontuação analisando o design de interface de um app criado pelo aluno e, com base na rubrica, o aluno pode identificar em quais aspectos o design visual do seu aplicativo pode ser melhorado (Solecki, 2020). A rubrica utilizada no CodeMaster v1.0 foi também avaliada em termos da sua confiabilidade e validade demonstrando uma boa confiabilidade (sendo medida por meio do coeficiente alfa de Cronbach que alcançou um resultado satisfatório de $\alpha = 0,84$) e validade (com a maioria dos itens apresentando bons parâmetros de inclinação via Teoria de Resposta ao Item) (Solecki *et al*., 2020).

Porém, desde que este modelo de avaliação foi criado, os guias de estilo evoluíram e o Material Design passou por diversas atualizações. Assim, a proposta deste trabalho é que seja realizado um aprimoramento do modelo de avaliação, incluindo a rubrica e consequentemente a ferramenta CodeMaster UI Design v1.0, para incluir as evoluções dos guias de estilo e complementar a avaliação. Como resultado, espera-se fornecer uma ferramenta atualizada que traz auxílio aos professores e alunos da educação básica na avaliação do design de interface de projetos. Com isso espera-se facilitar ainda mais o ensino e popularização de computação e design de interfaces no Brasil.

2. Metodolodia

A metodologia de pesquisa utilizada neste trabalho é dividida em quatro etapas. A Etapa 1 é dedicada à fundamentação teórica, que envolve a análise dos conceitos sobre os temas abordados. A Etapa 2 foca no estado da arte, onde se realiza um mapeamento sistemático para identificar e analisar modelos de avaliação de design de interface para apps criados com App Inventor. As atividades incluem definir o protocolo de revisão, buscar e selecionar artigos relevantes, e extrair e analisar dados para criar um panorama dos modelos existentes, seguindo o processo descrito por Petersen et al. (2015). Na Etapa 3, o objetivo é a evolução do modelo de avaliação. Baseado na fundamentação teórica e no estado da arte, esta etapa inclui a análise dos problemas comuns no design de interfaces com App Inventor e a evolução da rubrica de avaliação. A Etapa 4 se concentra na evolução da ferramenta. Utilizando um processo iterativo de

engenharia de software, a ferramenta CodeMaster UI Design é ajustada conforme as mudanças na rubrica e são realizados testes de corretude para assegurar a funcionalidade da ferramenta atualizada.

3. Estado da arte

O estado da arte é levantado por meio de um mapeamento sistemático seguindo o procedimento proposto por Petersen et al. (2015). Visa-se a responder à seguinte pergunta: Quais meios existem para a avaliação de design de interface de aplicativos móveis criados com App Inventor no contexto educacional?

Observando de forma geral uma falta de soluções propostas especificamente para projetos App Inventor e/ou no contexto educacional, são consideradas também abordagens de avaliação de aplicativos móveis em geral e fora do contexto educacional. As 200 publicações mais relevantes, ao termo de pesquisa, encontradas em todas as bases de dados foram analisadas e subsequentemente filtradas para atender os critérios de inclusão definidos anteriormente. No total, foram encontradas nove publicações contendo algum tipo de avaliação automatizada do design de interface de aplicativos móveis Android, iOS ou que foram desenvolvidos com App Inventor. Das únicas duas publicações que apresentam abordagens educacionais, ambas tratam da mesma ferramenta (Solecki *et al.*, 2019)(Solecki *et al.*, 2020). Dentre as abordagens das publicações, 5 de 9 envolvem o uso de *Machine Learning* como solução para avaliação automatizada de UIs de maneira geral, treinadas com conjuntos de dados de *screenshots* de aplicativos móveis. A grande maioria dos casos (7 de 9) utilizou técnicas estatísticas para analisar sua performance.

Os resultados demonstram que existem abordagens de avaliação automatizada de design de interfaces móveis como um todo, embora sejam poucas. Entretanto, fica evidente a ausência de abordagens que abrangem o contexto educacional.Embora este mapeamento não tenha sido limitado apenas a propostas que avaliam aplicativos criados com App Inventor, foi encontrada somente uma abordagem educacional que foca neste tipo de projeto (Solecki *et al*., 2019)(Solecki *et al*., 2020) com base em uma rubrica de avaliação.

4. Rubrica “CodeMaster UI Design v. 2.0 - App Inventor”

O objetivo deste trabalho é evoluir e complementar a rubrica CodeMaster UI Design – App Inventor v1.0 e a implementação e avaliação destas melhorias. A rubrica foi revisada em relação aos erros típicos encontrados (Tabela 14), a versão atual do Material Design (MD3) e fatores identificados na literatura. Na evolução da Rubrica v1.0 da ferramenta CodeMaster UI Design são mantidos alguns critérios e entram novos critérios e níveis de desempenho não abordados anteriormente. Além disso, alguns critérios e níveis de desempenho mantidos foram alterados de acordo com as orientações do Material Design 3 para continuar em conformidade com a nova versão. Como a avaliação será automatizada a partir do arquivo .aia, alguns dos erros identificados não foram expressados como critérios. O arquivo .aia não fornece alguns tipos de informações, como por exemplo informações sobre a posição dos elementos de forma explícita. Dessa forma, alguns erros, como em relação a simetria, não são

avaliados nesta versão da rubrica e outros, como alinhamento, são avaliados de forma simplificada.

Tabela 1. Rubrica CodeMaster UI Design v. 2.0

Nome da métrica no CodeMaster	Critério	≤0 pt	1 pt	2pt
Layout				
Densidade de tela	Qual é o número mínimo e o número máximo de elementos? (L4)	min ≤ 1 ou max > 40	min > 30 e max ≤ 40	min ≥ 1 e max ≤ 30
Espaço entre componentes		Nao usou espaço em branco (label ou alinhamento vazio) entre elementos		Usou pelo menos um espaço em branco (label ou alinhamento vazio) entre elementos
Formato dos botões	Todos os botões têm forma arredondada?	Uso de botão oval	Uso de botão retangular	Uso de botão arredondado/padrão
Tamanho dos componentes alvos de toque	Todos os componentes de “touch target” têm largura e altura maiores ou iguais a 10 pixels? (L1)	Não		Sim
Tamanho consistente de botões	Os botões agrupados na interface têm sempre o mesmo tamanho (altura e largura)? (L3)	Não possuem ou possuem em menos de 75% dos casos		Possuem o mesmo tamanho em 75% ou mais dos casos
Tipografia				
Familia da fonte	A família de fontes utilizada é com serifa? (T1)	Sim		Não
Uso de itálico	Tem algum texto em itálico? (T4)	Sim		Não
Uso de negrito	Existe algum parágrafo de texto (mais de 3 palavras) todo em negrito?	Em menos de 50% dos casos	Entre 50% e 80% dos casos	≥ 80% dos casos
	Algum botão tem texto em negrito? MD3: Fonte do botão: Label large Roboto Medium 14	Sim		Não
Tamanho da fonte de botões	Todos os botões com texto têm tamanho de fonte entre 12 a 24? (T2)	Não		Sim
Tamanho da fonte de componentes	Todos os componentes com texto (exceto botões) usam tamanhos entre 10px e 96px?	Não		Sim
Botões com texto não centralizado	Existe algum botão com texto não centralizado?	Sim		Não
Quantidade de fontes	Existem mais de 2 fontes diferentes usadas no aplicativo?	4 ou mais fontes diferentes	3	1-2 fontes
Escrita				
Capitalização de botões	Todos os botões tem legenda que começam com	Em menos de	Entre 50%	≥ 80%

	letra maiúscula? MD3: Legendas de botões devem começar com letra maiúscula e as demais serem minúsculas (W1)	50% dos casos	e 80% dos casos	dos casos
Capitalização de sentenças	Todas as frases começam com uma letra maiúscula ou um dígito? (W2)	Em menos de 50% dos casos	Entre 50% e 80% dos casos	≥ 80% dos casos
Texto padrão dos componentes	Todos os componentes têm um texto diferente do padrão (por exemplo, "*Text for Button1*")? (W3)	Em menos de 80% dos casos		≥ 80% dos casos
Evitar dois pontos	Existem legendas que terminam com ":" (dois pontos)? (W4)	Sim		Não
Evitar ponto final	Existem frases "não finais" terminando com "." (W5)	Em menos de 50% dos casos	Entre 50% e 80% dos casos	≥ 80% dos casos
Comprimento do texto de botões	A maior legenda de botão tem quantos caracteres? (W6)	Mais de 24		24 ou menos
	Cores			
Sistema de cores	Quantas cores são usadas (além de vermelho, preto, branco e cinza)? (C1) MD3: 5 cores (tonalidade) + vermelho para erro + preto e branco	7 ou mais, ou nenhuma	6	1 a 5
Uso de cores agradáveis	Existe alguma cor com saturação acima de 70% e brilho acima de 80%?	Sim		Não
Uso de cores harmônicas	As tonalidades de cores usadas são harmônicas entre si (monocromáticas, complementares, análogas, quadráticas, meio complementares ou triádicas)? (C4)	Não		Sim
Fundo dos labels transparentes	Existem labels com uma cor de fundo não transparente?	Sim		Não
Contraste entre texto e cor de fundo	Qual é o nível WCAG do aplicativo em relação ao contraste do texto? (C2) MD3: Texto grande (18pts ou maior, ou 14pts ou maior se estiver em negrito) deve atender a uma taxa de contraste de 3:1 para acessibilidade padrão (classificada como AA) e 4.5:1 para maior acessibilidade (classificada como AAA); Os tamanhos padrão do corpo do texto (menores que 18pts ou menores que 14pts se estiver em negrito) devem atender a uma taxa de contraste de 4.5:1 para acessibilidade padrão (classificada como AA) e 7:1 para maior acessibilidade (classificada como AAA). (espaços de cores HSL, HCL ou HCT comparando luminosidade ou tom)	Menos de 80% dos componentes são pelo menos Nível AAA	Mais de 80% dos componentes são pelo menos Nível AAA	Todos os componentes são pelo menos Nível AAA
Botões com mesma cor de fundo	Existem botões preenchidos com cores diferentes?	Mais de 4 cores		Até 4 cores
	Imagens e ícones			
Pixelização	Existem imagens pixeladas?	Sim		Não

5. Implementação da rúbrica

As melhorias sugeridas para a ferramenta CodeMaster foram desenvolvidas mantendo o modelo arquitetônico da versão existente. Para a evolução do CodeMaster v1.0, utiliza-se a linguagem de programação Java no backend integrada à *framework* Spring.

Para a implementação da rubrica CodeMaster UI Design – App Inventor, foram realizadas diversas alterações no módulo RESTGrader para incorporar a nova rubrica. Primeiramente, foram criadas métricas para botões com texto não centralizado, cores dos botões, fundo dos labels, quantidade de fontes, cores agradáveis e espaço entre componentes, conforme detalhado na tabela 15. Em seguida, foram ajustadas as métricas de capitalização de botões, capitalização de sentenças, formato dos botões, sistema de cores, uso de negrito, texto padrão dos componentes, tamanho consistente dos botões, evitar ponto final e comprimento do texto dos botões. No caso das métricas relacionadas à capitalização, tamanho dos botões, uso de negrito, texto padrão, ponto final e texto dos botões, as avaliações foram flexibilizadas, permitindo uma tolerância de até 80% de conformidade, especificada na Tabela 15, para não prejudicar as notas dos aplicativos de referência.

Adicionalmente, foram removidas as métricas de cores do material design, ícones do material design e distorção de imagens. As métricas referentes às cores e ícones, implementadas no contexto do Material Design 2, foram removidas por não serem mais aplicáveis no contexto do Material Design 3. A métrica de distorção de imagens, composta por três submétricas, foi removida por não apresentar os resultados esperados.

Foi aplicado um peso 4 em métricas específicas, cujos resultados são prioritários para determinar a conformidade da UI, sendo estas: componentes próximos à borda, uso de negrito, espaço entre componentes, uso de itálico, capitalização de botões, evitar dois pontos, contraste entre texto e cor de fundo, sistema de cores, fundo dos labels, cores harmônicas e cores agradáveis. O cálculo das médias das categorias e da média final da interface foi ajustado para não contabilizar métricas avaliadas como N/A. Além disso, foi aplicado o modelo de arredondamento da UFSC na média final da interface, onde o valor é arredondado em intervalos de 0,5.

6. Avaliação de concordância do modelo de avaliação

Com o objetivo de avaliar a qualidade da nova versão da rubrica proposta foi realizada uma avaliação da concordância da nota gerada automaticamente pelo modelo de avaliação implementado em comparação com uma nota alocada por um modelo de avaliação da estética visual desenvolvido por Lima (2022). Para esta avaliação foram selecionados 5 aplicativos criados com App Inventor no contexto da iniciativa Computação na Escola/INCoD/INE/UFSC e/ou disponibilizados na App Inventor Gallery, com designs de interface de usuários bons e ruins. Cada um dos aplicativos foi avaliado pelo sistema implementado do modelo evoluído Codemaster - UI Design v2.0 e pelo modelo de avaliação da estética visual (Lima, 2022). As notas resultantes sao apresentados na Tabela 2.

Tabela 2. Notas do CodeMaster v. 2.0 e do modelo de avaliação de estética visual

	Controle o seu peso	**Covidhope**	**Smart Medicine**	**Xô Dengue**	**Yu**
Nota CodeMaster	5,5	4	4,5	9,5	10

UI Design 2.0					
Avaliação Estética (Lima, 2022)	4,6	4,25	5,4	6,14	7,6

Para avaliar a concordância entre as notas geradas pelo CodeMaster UI Design 2.0 e o modelo de avaliação de Lima (2022), foram utilizados os seguintes métodos estatísticos: Intraclass Correlation Coefficient (modelo ICC3) e o gráfico de Bland-Altman.

O Intraclass Correlation Coefficient (ICC) é uma medida estatística utilizada para avaliar a consistência ou concordância entre diferentes avaliadores. O modelo ICC3 é apropriado para esta análise porque todos os 5 aplicativos foram avaliados por ambos os avaliadores (CodeMaster e Modelo de estética visual). A análise do coeficiente de correlação intraclasse (ICC3) resultou em um valor de 0,6791. De acordo com a classificação de Cichetti (1994), correlações entre 0,75 e 1,0 são consideradas excelentes; de 0,6 a 0,75 são boas; de 0,4 a 0,6 são razoáveis; e menores que 0,4 são consideradas pobres. Portanto, o valor obtido de 0,6791 indica uma correlação boa entre as avaliações.

A análise de Bland-Altman revelou uma diferença média de 1,102, indicando um viés em que, em média, o Avaliador 1 atribui notas 1,102 unidades mais altas que o Avaliador 2. Os limites de concordância foram de -2,3849 (inferior) a 4,5889 (superior). Essa ampla variabilidade sugere que os critérios de avaliação podem não estar suficientemente padronizados.

7. Conclusão

Os resultados indicam que o Codemaster UI Design v2.0 consegue avaliar a qualidade do design de interfaces de apps App Inventor com uma boa correlação, mas ainda precisa de ajustes para alinhar suas avaliações com as percepções humanas de estética. As divergências entre as notas absolutas atribuidas pelos avaliadores podem ser justificadas por alguns fatores. Devido a limitações intrínsecas ao App Inventor, não foi possível avaliar alguns critérios importantes que influenciam significativamente a estética do aplicativo. utra justificativa é que a avaliação estética e o CodeMaster não avaliam a necessariamente a mesma coisa, sendo a avaliação estética bem mais ampla.

O aplicativo "Controle o seu peso" recebeu uma nota total de 5,5, superior ao esperado em comparação com as notas de estética, pois atende a muitos critérios estabelecidos, apesar de sua estética visual pouco atraente. "Smart Medicine" também apresentou falhas em alinhamento e simetria, mas ainda assim recebeu nota 6 em layout. O aplicativo “Yu” obteve nota 10 no CodeMaster e 7 na avaliação estética, mostrando boa atratividade visual, mas com falhas na avaliação de imagens e ícones.

Para melhorar, sugere-se a inclusão de novos critérios de layout e a implementação de um modelo de deep learning para avaliar esses critérios, além de

mecanismos para avaliar a qualidade de ícones e a conformidade das cores, o que pode aprimorar a qualidade da avaliação visual dos aplicativos.

Esta evolução do modelo de avaliação de abordagem automatizada pode fornecer aos alunos uma avaliação objetiva, minimizar o esforço dos professores e auxiliar aqueles sem formação específica em computação. Assim, 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 aprimorada tornando a avaliação de layout mais precisa com a adição de critérios de alinhamento, regularidade, simetria e balanceamento. Outra melhoria pode ser na avaliação de imagens, incorporando a capacidade de reconhecer características dos ícones, como *watermark* de licença e cores.

Referências

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

De Luca, C., Klinger, D. A. (2010). Assessment literacy development: Identifying gaps in teacher candidates' learning. *Assessment in Education: Principles, Policy & Practice, 17*.

Ferreira, M. N. F., Gresse von Wangenheim, C., Missfeldt Filho, R., Da Cruz Pinheiro, F., Hauck, J. C. R. (2019). Learning user interface design and the development of mobile applications in middle school. *ACM Interactions, 26*(4).

Lima, A., Martins, O. P. H. R., Gresse von Wangenheim, C., Von Wangenheim, A., Borgatto, A. F., Hauck, J. C. R. (2022). Automated assessment of visual aesthetics of Android user interfaces with deep learning. In *Proceedings of the Brazilian Computing Society* (pp. 1-10).

Oliveira, M., De Souza, A., Ferreira, A., Barreiros, E. (2014). Ensino de lógica de programação no ensino fundamental utilizando o Scratch: Um relato de experiência. *Workshop sobre Educação em Computação. Sociedade Brasileira de Computação* (pp. 239-248).

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology, 64*, 1-18.

Salim, F., Haque, U. (2015). 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, 81*, 31-48.

Solecki, I. S. S. (2020). Uma abordagem para a avaliação do design visual de aplicativos móveis criados com linguagens de programação baseadas em blocos (Master's thesis, Universidade Federal de Santa Catarina).

Solecki, I., Porto, J. A., Alves, N. d. C., Gresse von Wangenheim, C., Hauck, J. C. R., Borgatto, A. F. (2020). Automated assessment of the visual design of Android apps developed with App Inventor. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 1-6). Portland, USA.

Valentim, N. M. C., Silva, W. A. F., Conte, T. (2015). Avaliando a experiência do usuário e a usabilidade de um aplicativo web móvel: um relato de experiência. In *Proceedings of the XVIII Congresso Ibero-Americano em Engenharia de Software* (pp. 1-5). Lima, Peru.