UNIVERSIDADE FEDERAL DE SANTA CATARINA

Matheus Faustino Demetrio

Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação

Florianópolis – SC

2017/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Matheus Faustino Demetrio

Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências Da Computação.

Orientador: Prof. Dr. rer. nat. Christiane A. Gresse von Wangenheim, PMP

Co-Orientador: Prof. Dr. Jean Carlo Rossa Hauck

UFSC
Florianópolis

Matheus Faustino Demetrio

Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção de grau de Bacharel em Ciências Da Computação.

Orientador(a):
Prof. Dr. rer. nat. Christiane A. Gresse von Wangenheim, PMP

_____

Co-Orientador(a):
Prof. Dr. Jean C. R. Hauck

_____

Banca examinadora:
Giselle Araújo e S. de Medeiros

_____

RESUMO

Com a evolução da tecnologia e a sua inserção em todas as áreas profissionais e pessoais dos cidadãos, percebe-se a necessidade de incluir o ensino do pensamento computacional no Ensino Básico. O pensamento computacional envolve o raciocínio lógico e matemático, desenvolvimento e análise de algoritmos, representação de dados e técnicas de programação para a solução de problemas. Tipicamente o pensamento computacional é ensinado por meio de unidades instrucionais que envolvem trabalhos práticos de programação utilizando ambientes de programação para iniciantes. Um exemplo é o *App Inventor,* um ambiente de programação baseado em blocos que possibilita o desenvolvimento de apps para dispositivos móveis Android. Neste contexto educacional, para facilitar a avaliação dos trabalhos práticos de programação é importante que o professor tenha auxilio por meio de ferramentas. Deste modo, o objetivo deste trabalho é desenvolver uma ferramenta que analisa e avalia automaticamente o código desenvolvido pelos alunos usando o App Inventor, auxiliando os professores na avaliação dos trabalhos práticos realizados. O analisador e avaliador de código é desenvolvido com base na literatura seguindo uma abordagem sistemática de desenvolvimento de software. Por meio da disponibilidade do analisador e avaliador de código, espera-se facilitar e reduzir o esforço necessário para avaliação de trabalhos de alunos no ensino de computação no Ensino Básico.

Palavras-chave: ensino da computação; programação; análise de código; App Inventor.

LISTA DE FIGURAS

Figura 1 - Modelo ADDIE (BRANCH, 2009) 14
Figura 2 - Níveis CSTA (CSTA, 2011) 18
Figura 3 - Áreas do ensino de computação no Ensino Básico (CSTA, 2011) 18
Figura 4 - Processo de análise estática (STRIEWE et al., 2014) 23
Figura 5 - Visão do fluxo do verificador adaptado voltado ao ensino (Sinthsirimana et al., 2013) 24
Figura 6 - Tela Designer do App Inventor (APP INVENTOR, 2017) 29
Figura 7 - Tela Blocos do App Inventor (APP INVENTOR, 2017) 30
Figura 8 - Tela Inicial do Dr. Scratch 40
Figura 9 - Tela de Resultado de Análise do Dr. Scratch 40
Figura 10 - Tela Inicial do Ninja Code Village 44
Figura 11 - Tela de Resultado de Análise do Ninja Code Village 44
Figura 12 - Diagrama de Casos de Uso do CodeMaster 55
Figura 13 - Diagrama de Componentes do CodeMaster 58
Figura 14 - Diagrama de Classes do Serviço REST 60
Figura 15 - Diagrama de Classes do analisador e avaliador do App Inventor 61
Figura 16 - Sequência da Análise de Código CodeMaster 62
Figura 17 - Diagrama de Classes Reduzido da Análise de Código do Módulo de Apresentação 64
Figura 18 - Diagrama de Classes do Cadastro de Professor 65
Figura 19 - Diagrama de sequência de upload de múltiplos projetos 66
Figura 20 - Diagrama Entidade Relacionamento do Banco de Dados CodeMaster 67
Figura 21 - Resultados da avaliação manual de projetos App Inventor 81
Figura 22 - Avaliação pelo CodeMaster de Projetos App Inventor 81
Figura 23 - Fotos de alguns avaliadores do CodeMaster 84
Figura 24 - Quantidade de professores por disciplina 84
Figura 25 - Quantidade de alunos por ano do ensino 85

LISTA DE TABELAS

Tabela 1 - Objetivos de aprendizagem de Programação, nível 2 (CSTA, 2011) 20
Tabela 2 - Componentes do Designer 29
Tabela 3 - Blocos de Controle 31
Tabela 4 - Blocos Lógicos 31
Tabela 5 - Blocos Matemáticos 32
Tabela 6 - Blocos de texto 32
Tabela 7 - Blocos de Lista 33
Tabela 8 - Blocos de Cores 33
Tabela 9 - Blocos de Variáveis 34
Tabela 10 - Blocos de Procedimentos 34
Tabela 11 - Termos de Busca 35
Tabela 12 - Strings de Busca por Base 36
Tabela 13 - Resultado da Execução da Busca 38
Tabela 14 - Analisadores de Código Relevantes Encontrados 39
Tabela 15 - Rubrica Dr. Scratch (DR. SCRATCH, 2016) 41
Tabela 16 - Comparação Entre Analisadores em Relação as Suas Funcionalidades/Características 45
Tabela 17 - Conceitos do Programação Analisados pelos Analisadores 46
Tabela 18 - Conceitos do Programação Analisados pelo CodeMaster e Rubrica de Avaliação 49
Tabela 19 - Mapeamento Nota e Faixa Ninja 51
Tabela 20 - Requisitos Funcionais 52
Tabela 21 - Requisitos Não Funcionais 54
Tabela 22 - Interfaces gráficas do USC01 72
Tabela 23 - Interfaces gráficas do USC02 73
Tabela 24 - Interfaces gráficas do USC03 74
Tabela 25 - Interfaces gráficas do USC04 75
Tabela 26 - Interfaces gráficas do USC05 77
Tabela 27 - Projetos App Inventor selecionados para avaliação da corretude 79
Tabela 28 - Fatores de qualidade analisados 82
Tabela 29 - Análise da Utilidade do CodeMaster por alunos 85
Tabela 30 - Análise da Funcionalidade do CodeMaster por alunos 87
Tabela 31 - Análise de desempenho do CodeMaster 88
Tabela 32 - Análise da facilidade de uso do CodeMaster 88
Tabela 33 - Pontuações de Usabilidade do CodeMaster 89

SUMÁRIO

1. INTRODUÇÃO 7
1.1 CONTEXTUALIZAÇÃO 7
1.2 OBJETIVOS 9
1.3 METODOLOGIA DE PESQUISA 10
1.4 ESTRUTURA DESTE DOCUMENTO 12
2. FUNDAMENTAÇÃO TEÓRICA 13
2.1 APRENDIZAGEM E ENSINO 13
2.2 ENSINO DE COMPUTAÇÃO NO ENSINO FUNDAMENTAL II 16
2.3 ANÁLISE E AVALIAÇÃO DE CÓDIGO 21
2.3.1 ANÁLISE DE CÓDIGO APLICADA À EDUCAÇÃO 25
2.3 APP INVENTOR 27
3. ESTADO DA ARTE 35
3.1 DEFINIÇÃO DO PROTOCOLO DE REVISÃO 35
3.2 EXECUÇÃO DA BUSCA 37
3.3 ANALISE DOS TRABALHOS ENCONTRADOS 39
3.3.1 DR. SCRATCH 39
3.3.2 NINJA CODE VILLAGE FOR SCRATCH 42
3.4 DISCUSSÃO 45
3.4.1 AMEAÇAS A VALIDADE DO MAPEAMENTO SISTEMÁTICO 46
4. DESENVOLVIMENTO DO CODEMASTER – APP INVENTOR 48
4.1 MODELO CONCEITUAL 48
4.2 ANÁLISE DOS REQUISITOS 52
4.2.1 REQUISITOS FUNCIONAIS 52
4.2.2 REQUISITOS NÃO-FUNCIONAIS 54
4.3 CASOS DE USO 55
4.4 MODELAGEM E IMPLEMENTAÇÃO DO CODEMASTER 58
4.4.1 ARQUITETURA DO MÓDULO DE AVALIAÇÂO 59
4.4.2 ARQUITETURA DO MÓDULO DE APRESENTAÇÃO 63
4.4.3 TESTES DE UNIDADE DO SOFTWARE 68
4.4.4 EXPANDINDO O CODEMASTER 69
4.4.5 DESIGN DE INTERFACE DO CODEMASTER 71
5. AVALIAÇÃO DO CODEMASTER – APP INVENTOR 78
5.2 AVALIAÇÃO DA CORRETUDE 78
5.2.1 EXECUÇÃO DA AVALIAÇÃO DA CORRETUDE 78
5.3 AVALIAÇÃO POR USUÁRIOS 82
5.3.1 DEFINIÇÃO DA AVALIAÇÃO POR USUÁRIOS 82
5.3.2 EXECUÇÃO DA AVALIAÇÃO POR USUÁRIOS 83
5.3.3 ANÁLISE DOS DADOS 85
6. CONCLUSÃO 91
REFERÊNCIAS 93

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Os computadores transformaram completamente o mundo e o mercado de trabalho (MERCADO, 2002). Juntamente com estas mudanças, a computação e as tecnologias advindas delas fazem parte do coração da economia mundial e do modo de vida das pessoas. Atualmente todos os estudantes possuirão uma vida altamente influenciada pela computação e muitos trabalharão em áreas em que a computação está diretamente envolvida (CSTA, 2011). A computação é a busca por soluções para um determinado problema, dada uma determinada entrada, obter um resultado por meio de um algoritmo. Ela envolve o entendimento, o desenvolvimento e projeto de computadores, assim como processos computacionais (CSTA, 2011).

Portanto todos os cidadãos devem possuir um entendimento básico sobre os princípios e práticas da computação, podendo se tornar criadores de sistemas computacionais e também consumidores mais competitivos e produtivos (CSTA, 2011).

A falta de incentivo no ensino da computação nas escolas de ensino básico acarreta em dois principais problemas: o não despertar do interesse dos jovens pela área da computação e o não mostrar o que realmente é ensinado nos cursos superiores para atrair o público alvo correto. Isso acaba gerando um outro grande problema, carência de mão-de-obra qualificada nas empresas de Tecnologia da Informação e Comunicação (ACATE, 2012). Além de influenciar os cidadãos que precisam deste conhecimento nas suas vidas profissionais e pessoais.

Por esses motivos é importante que o ensino da computação seja incluído no Ensino Básico, contribuindo desta maneira na popularização da computação na sociedade em geral. Contudo, isso atualmente não acontece, geralmente não é dada a mesma atenção a computação quanto a outras disciplinas clássicas. Mesmo abordando o ensino do uso de TI (p.ex. o uso de editores de texto, editores de imagem, etc.) atualmente, na maioria dos casos, não se ensina o pensamento computacional, abordando conceitos como a representação de dados, criação de algoritmos e análise, examinação de diferentes algoritmos baseando-se em outras estratégias de solução, que são conceitos dos quais devem ser ensinados já aos alunos no Ensino Básico (CSTA, 2011).

Por outro lado, os alunos do ensino básico já possuem uma interação muito forte com os dispositivos existentes, como *smartphones*, *tablets* e computadores. Cerca de 82% das crianças do Brasil acessam a internet de dispositivos móveis (BARBOSA, 2015), facilitando algumas iniciativas existentes focadas em levar o ensino da computação para crianças do mundo inteiro (*HOUR OF CODE*, 2016) (COMPUTAÇÂO NA ESCOLA, 2016). Portanto, deve-se usar isto em benefício do ensino, usar os dispositivos que fazem parte do seu dia a dia, que os alunos já estão acostumados, e focar o ensino de computação por meio do desenvolvimento de aplicativos móveis.

Isto pode ser feito utilizando o App Inventor. O App Inventor (APP INVENTOR, 2016) é um ambiente de programação visual baseado em blocos que permite que qualquer pessoa, inclusive crianças, comecem a programar e construir aplicativos completos para dispositivos Android. Para a programação destes aplicativos os alunos aprendem a usar diversos tipos de comandos como laços, condicionais e procedimentos, iguais aos usados na programação convencional por texto, e então, indiretamente, o pensamento computacional.

Porém, mesmo com a disponibilidade aberta do App Inventor, a programação de apps ainda não é ensinada amplamente nas escolas. Uma das barreiras à integração do ensino de computação nas escolas é a falta de ferramentas que dão suporte aos professores na avaliação de trabalhos práticos de programação dos estudantes (MORENO-LEÓN et al., 2015). Esta situação se agrava ainda, pelo fato de que atualmente existe uma falta de professores do Ensino Básico com formação para o ensino de computação, sendo ensinada, tipicamente, por professores de outras disciplinas, como história por exemplo, de forma interdisciplinar.

Levando em consideração especialmente a falta de formação dos professores do Ensino Básico, a avaliação dos programas de aplicativos desenvolvidos pelos alunos representa uma atividade complexa e de um esforço considerável para os professores (ESERYEL et al., 2013). Nesta situação, um avaliador automático de código que permite uma análise e avaliação automatizada do código desenvolvido pelos alunos pode facilitar esta parte, reduzindo a necessidade de competências específicas do professor e também reduzindo o esforço necessário para analisar todos os programas desenvolvidos pelos alunos. Uma análise automatizada pode também aumentar a objetividade e eliminar qualquer favoritismo ou inconsistência (ZEN et al., 2011).

1.2 OBJETIVOS

OBJETIVO GERAL

O trabalho tem como objetivo desenvolver uma ferramenta web para analisar e avaliar código desenvolvido no App Inventor, a ser utilizado para avaliar trabalhos práticos no ensino de programação de aplicativos móveis no Ensino Básico (no Ensino Fundamental II focando em jovens de 11 a 14 anos de idade) alinhado à referência de curriculo (CSTA, 2016).

OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

O1. Sintetizar a fundamentação teórica sobre aprendizagem e ensino de computação no Ensino Básico, o ambiente App Inventor, a análise de código e avaliação no processo de ensino.

O2. Analisar o estado da arte sobre analisadores de código referentes a linguagens de programação baseada em blocos.

O3. Desenvolver uma ferramenta web para a análise de código desenvolvido no App Inventor.

O4. Aplicar e avaliar a ferramenta desenvolvida na prática.

Premissas e restrições: O projeto é realizado de acordo com o regulamento vigente do Departamento de Informática e Estatística (INE/UFSC) em relação aos Trabalhos de Conclusão de Curso. A ferramenta tem como foco somente a avaliação de aplicativos desenvolvidos com o App Inventor. Estão fora deste contexto outros ambientes de ensino de computação para crianças.

O presente trabalho é desenvolvido em paralelo com o desenvolvimento do Trabalho de Conclusão de Curso do aluno Rafael Pelle (PELLE, 2017), que por sua vez tem como objetivo, também, criar uma ferramenta web para a análise de código, porém, para outro ambiente de programação: Snap!. Deste modo, o esforço de desenvolvimento da parte estrutural de servidor e interface gráfica do sistema é dividida, permitindo criar uma estrutura única, mais robusta, funcional e expansível. O desenvolvimento do projeto da interface gráfica foi auxiliado pelos alunos Heliziane

Barbosa e Luiz Felipe Azevedo, bolsistas da iniciativa Computação na Escola do GQS/InCod/INE/UFSC.

1.3 METODOLOGIA DE PESQUISA

Neste trabalho se adota uma abordagem multi-método. A metodologia de pesquisa utilizada neste trabalho é dividida em 4 etapas:

Etapa 1 – Análise da literatura referente ao ensino e aprendizagem, assim como computação em dispositivos móveis e aplicativos. Esta primeira etapa é dividida nas seguintes atividades:
A1.1 – Análise teórica sobre o ensino e aprendizagem de computação no Ensino Básico.
A1.2 – Análise teórica do ambiente de desenvolvimento App Inventor.
A1.3 - Análise teórica sobre aplicações de análise de código.

Etapa 2 – Levantamento do estado da arte de analisadores de código de ambientes de programação visual (baseado em blocos) como App Inventor, Snap!, Scratch. Nesta etapa é realizado um estudo de mapeamento (PETERSEN, et al., 2008). Essa etapa está dividida nas seguintes atividades:
A2.1 – Definição do protocolo de estudo.
A2.2 – Execução da busca de analisadores de código de ambientes de programação baseado em blocos.
A2.3 – Extração e análise das informações relevantes sobre analisadores de código e análise dessas informações potencializando-as para serem utilizadas no analisador a ser desenvolvido no trabalho.

Etapa 3 - Desenvolvimento uma ferramenta web para análise de códigos do App Inventor seguindo um processo de engenharia de software iterativo incremental (LARMAN & BASILI, 2003). Na primeira iteração são levantados os requisitos e modelagem da arquitetura do sistema. Na segunda iteração é desenvolvido o sistema de análise e avaliação de código do App Inventor. Na terceira iteração é implementado

em parceria com Pelle (2018) a integração dos analisadores de App Inventor e Snap! e a estrutura de servidor. Na quarta iteração a interface gráfica é desenvolvida em parceria com os bolsistas da iniciativa Computação na Escola, incluindo pesquisas sobre a interface gráfica com o público alvo. Por último são feitos os testes no sistema e aplicada possíveis correções e ajustes.

A3.1 - Análise de requisitos e modelagem da arquitetura do sistema.

A3.2 – Modelagem baixo nível e implementação do analisador e avaliador de códigos do App Inventor.

A3.3 – Modelagem baixo nível e implementação da estrutura de serviços do servidor e integração dos analisadores.

A3.4 – Desenvolvimento da interface gráfica do sistema.

A3.5 - Testes do sistema.

Etapa 4 – Avaliação e aplicação da ferramenta desenvolvida.

Nesta etapa a ferramenta desenvolvida é avaliada. Para isto são realizados dois tipos de avaliação. É realizada uma avaliação com o objetivo de avaliar a corretude do analisador de código comparando os resultados gerados automaticamente pela ferramenta com uma análise manual do código.

Outra avaliação é realizada em relação a aplicabilidade da ferramenta na prática educacional por meio de testes de usuário. Essa etapa é dividida nas seguintes atividades:

A4.1 - Avaliação da corretude.

A4.1.1 - Definir a avaliação da corretude.

A4.1.2 - Executar a avaliação da corretude com a ferramenta desenvolvida e de forma manual.

A4.1.3 - Analisar e comparar os resultados.

A4.2 - Avaliação da aplicabilidade.

A4.2.1 - Definir a avaliação por meio de testes de usuários.

A4.2.2 – Coletar os dados dos testes de usuários.

A4.2.3 – Analisar os dados coletados.

A4.3 - Discutir os resultados.

1.4 ESTRUTURA DESTE DOCUMENTO

Na seção 2 deste trabalho são abordados os conceitos da base do ensino e aprendizagem em geral e focados no ensino de computação. Além dos conceitos de análise e avaliação no contexto do Ensino Básico.

Na seção 3 é levantado o estado da arte sobre analisadores e avaliadores automatizados de código voltados a linguagens de programação visuais para o ensino de computação no Ensino Básico.

Na seção 4 é apresentado o desenvolvimento de uma ferramenta de análise e avaliação de código voltado a *apps* produzidos no App Inventor. Nesta seção é apresentado os passos da implementação, bem como as técnicas e tecnologias utilizadas para a criação da ferramenta.

Na seção 5 são abordadas as avaliações de corretude da ferramenta bem como as avaliações de utilidade, funcionalidade, desempenho e usabilidade medidas por meio de testes com especialistas da área do ensino de computação.

Na seção 6 são apresentadas as conclusões. São verificados se os objetivos propostos foram atendidos, qual a contribuição da ferramenta para a sociedade e propostas de futuros trabalhos.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos referentes a teoria da aprendizagem e do ensino, bem como o princípio de uma unidade instrucional e *design* instrucional. Também são apresentados conceitos de análise de código e uma visão geral sobre o ambiente de programação App Inventor.

2.1 APRENDIZAGEM E ENSINO

A aprendizagem é o processo de assimilação de conhecimentos, habilidades ou valores por meio do estudo, da prática, da experiência, do raciocínio, ou da observação. A aprendizagem está presente em qualquer atividade humana em que possamos aprender algo (LIBANEO, 1999). Aprendizagem pode acontecer de diversas formas, nas relações interpessoais do dia-a-dia ou sozinho, mas para aumentar as chances de a aprendizagem ser bem-sucedida, ela é institucionalizada por meio do ensino.

O ensino é o processo de transmissão e apropriação de uma competência (NEUNERET et al., 1981). A competência pode ser vista como sendo a junção de conhecimentos, de habilidades e de atitudes. No ensino, usualmente, existem como atores, o instrutor e o aprendiz. Assim, os objetivos da interação entre os atores, resulta no ato de aprender do aprendiz.

No ensino cria-se e usa-se unidades instrucionais (UIs) guiando e apoiando os instrutores (MARCONDES et al., 2009). Uma unidade instrucional pode ser uma aula, um curso, uma oficina, exercícios, atividades, organizado em uma sequência lógica, que guie de forma precisa e facilitadora o instrutor. A unidade instrucional apresenta uma visão concisa ao instrutor do objetivo da aprendizagem, do conteúdo e da sequência e os métodos de ensino a ser ensinado, o que inclui os materiais e ferramentas que serão utilizados (VON WANGENHEIM et. al., 2014).

Porém, para assegurar que a aprendizagem seja efetiva e eficiente e ao mesmo tempo motivadora e divertida, é importante que as unidades instrucionais sejam desenvolvidas e avaliadas seguindo um processo e sistemática de *design* instrucional (VON WANGENHEIM et. al., 2014).

Design instrucional é a "ação intencional e sistemática de ensino, que envolve o planejamento, o desenvolvimento e a utilização de métodos, técnicas, atividades, materiais, eventos e produtos educacionais em situações didáticas específicas, a fim de facilitar a aprendizagem humana a partir dos princípios de aprendizagem e instrução conhecidos" (FILATRO, 2004).

Existem diversos modelos de design instrucional, chamados de *Instructional System Design* (ISD) (BRANCH, 2009), que seguem, em sua maioria, como referência, o modelo ADDIE[1] (PISKURICH, 2015). O modelo ADDIE consiste em cinco etapas para a criação de unidades instrucionais (Figura 1).

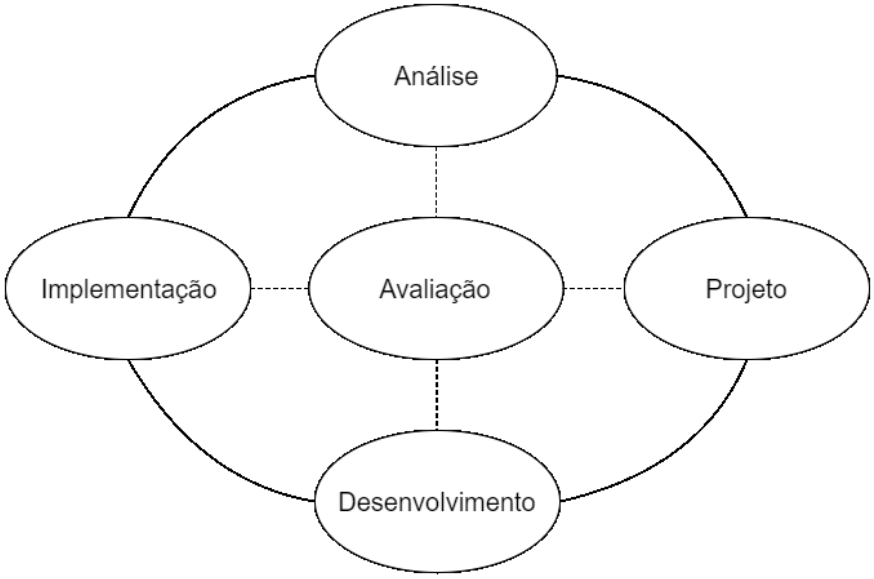


Figura 1 - Modelo ADDIE (BRANCH, 2009)

As etapas do modelo ADDIE são detalhadas a seguir conforme BRANCH (2009).

O propósito da etapa de **Análise** é identificar as necessidades de aprendizagem. É importante determinar as metas e os objetivos da unidade instrucional e analisar o público-alvo considerando informações pessoais dos alunos, como nacionalidade, idade, experiências educacionais e interesses. Se faz importante também, analisar e definir outros fatores da unidade instrucional como um todo, como recursos humanos e seus requisitos, recursos técnicos, infraestrutura, custo e tempo. Como resultado é criado a caracterização do contexto.

[1] ADDIE é um acrônimo para *Analyze, Design, Develop, Implement e Evaluate.*

Na etapa de **Projeto** deve-se especificar o objetivo geral e os objetivos de aprendizagem da UI. São definidos o conteúdo a ser abordado, o sequenciamento deste conteúdo, os métodos instrucionais que serão utilizados para especificar a natureza das atividades que alunos e professor estarão envolvidos durante a aula e os tipos de atividades que serão realizadas pelos alunos (colaborativas, interativas, individuais). Com base nestas, os materiais que serão utilizados na unidade instrucional são especificados.

Também são definidas, na etapa de Projeto, as avaliações a serem realizadas para acompanhar e avaliar o progresso de aprendizagem dos alunos. As avaliações dos alunos serão relacionadas aos objetivos de aprendizagem definidos. Como resultado é definido o plano de ensino.

A etapa de **Desenvolvimento** visa a criação dos materiais que serão utilizados durante a unidade instrucional. Dependentemente dos materiais identificadas na etapa anterior, desenvolve-se slides, vídeos, exercícios e/ou até sistemas de software (p.ex.: exemplos de apps) usando ferramentas diversas (papel, caneta, editores de texto, softwares, entre outros). Como resultado são desenvolvidos os materiais instrucionais. Esta etapa também pode envolver a seleção e/ou desenvolvimento de ferramentas de gerenciamento de UIs e outras ferramentas que suportam os processos instrucionais, como por exemplo, analisadores de trabalhos, incluindo analisadores de código.

O objetivo da etapa de **Implementação** é preparar o ambiente de aprendizagem e capacitação dos instrutores. Nesta etapa é aplicada a unidade instrucional na prática. Isso envolve a aplicação dos métodos instrucionais conforme definido no plano de ensino na etapa de projeto. Com a conclusão da etapa de implementação, deve haver um novo ambiente de aprendizagem, em que os alunos podem começar a construir novas competências. Esta etapa é um divisor, pois ela indica a conclusão de atividades de desenvolvimento e é considerada uma transição entre das avaliações formativas para as avaliações somativas.

A etapa de **Avaliação** tem o propósito de avaliar a qualidade da unidade instrucional e os seus processos, visando a melhoria continua da unidade instrucional. Para isso é necessário definir e conduzir a avaliação a ser realizada e o modo de coleta dos dados de forma que seja possível avaliar a unidade instrucional com confiabilidade e validade dos dados.

Voltado a avaliação do desempenho do aluno (e não da qualidade da UI), o objetivo é medir o grau com que ele atingiu o(s) objetivo(s) de aprendizagem. Existem

vários tipos de avaliações que podem ser feitas para medir o desempenho que o aluno atinge na aprendizagem (CORTESÃO, 2002). As avaliações podem ser provas (tanto escritas como orais), exercícios, trabalhos práticos, seminários, etc. As provas têm o objetivo de serem mais pontuais naquilo que querem avaliar, geralmente avaliando a parte mais teórica da aprendizagem, aumentando a precisão das medidas educacionais (HAYDT, 1988). Os trabalhos práticos abrangem mais o nível de aprendizagem de aplicação, e avaliam com mais foco a parte prática da aprendizagem.

No ensino da computação, é possível inferir as competências adquiridas durante o processo de aprendizagem por meio da análise/avaliação de trabalhos práticos de programação. Nos trabalhos é possível identificar se os alunos têm capacidade de colocar em prática os conceitos teóricos, mostrando habilidade em organizar, sintetizar e aplicar a informação e a competência adquirida, e a habilidade em gerenciar os recursos disponíveis (MORENO-LEÓN et al., 2015).

Porém, a análise destes trabalhos práticos na avaliação, tipicamente, requer um esforço considerável e competência do professor na área de conhecimento (MEDEIROS, 1974).

2.2 ENSINO DE COMPUTAÇÃO NO ENSINO FUNDAMENTAL II

O **Ensino Fundamental** é uma das etapas que compõem o que é chamado de Ensino Básico no Brasil, juntamente com o Ensino Infantil e Ensino Médio, e que segundo a nova Lei de Diretrizes e Bases da Educação Nacional (Lei Federal nº 9.394, 1996), tem por finalidade “desenvolver o educando, assegurar-lhe a formação indispensável para o exercício da cidadania e fornecer-lhe meios para progredir no trabalho e em estudos posteriores”. O Ensino Fundamental é organizado em duas etapas, a primeira etapa engloba os 5 primeiros anos, chamado Ensino Fundamental I ou Anos Iniciais, e a segunda etapa engloba os quatro anos finais, do 6º ao 9º ano, chamado Ensino Fundamental II ou Anos Finais (MINISTÉRIO DA EDUCAÇÃO, 2005). Idealmente os alunos devem entrar no ensino fundamental aos seis anos de idade e sair aos quatorze.

Enfocando o Fundamental II, foco do presente trabalho, os objetivos e conteúdo do Ensino Fundamental II são definidos por Parâmetros Curriculares Nacionais

(PCNs) que formam uma base comum para os currículos nacionais. De acordo com as PCNs, os professores devem orientar os seus alunos do 6º ao 9º ano de forma clara e objetiva, adequando o ensino aos ideais de democracia e buscando a melhoria da qualidade do ensino nas escolas brasileiras (OLIVEIRA, 2013). Além disso a PCN "[...] buscará eleger, como objeto de ensino, conteúdos que estejam em consonância com as questões sociais que marcam cada momento histórico" (PCN, 1998). As disciplinas que as PCNs abrangem são: Língua Portuguesa, Matemática, Ciências Naturais, Geografia, História, Arte, Educação Física, Língua Estrangeira e temas adicionais como Pluralidade Cultural, Meio Ambiente, Saúde e Orientação Sexual. Contudo, a computação é tratada como ensino de informática, resumindo-se ao ensino do uso de ferramentas básicas para dar apoio a informatização das demais disciplinas.

Porém, de acordo com a Sociedade Brasileira de Computação é importante que a computação seja inserida no contexto do ensino fundamental:

> "É importante salientar que devemos primar pela qualidade do ensino em todos os níveis da cadeia de formação de recursos humanos. Entendemos que a Computação deva ser ensinada desde o ensino fundamental, a exemplo de outras ciências como Física, Matemática, Química e Biologia. Esses são pontos muito importantes para que no futuro tenhamos recursos humanos qualificados para enfrentar os desafios que advirão (SBC, 2015)."

No Brasil, atualmente, existe uma proposta inicial de currículo criada pela Sociedade Brasileira de Computação (SBC, 2017) que trata as competências e habilidades que compõem a computação. Internacionalmente existem vários currículos que servem de referência para o Ensino Básico, entre eles um dos currículos de referência mais reconhecidos é o CSTA/ACM K-12 (CSTA, 2011) criado pela *Computer Science Teacher Association* e *Association for Computing Machinery* (ACM). Como as diretrizes CSTA para ensino da computação abrangem desde a educação infantil ao ensino médio, estas são organizadas em 3 níveis distintos conforme apresentado na Figura 2.

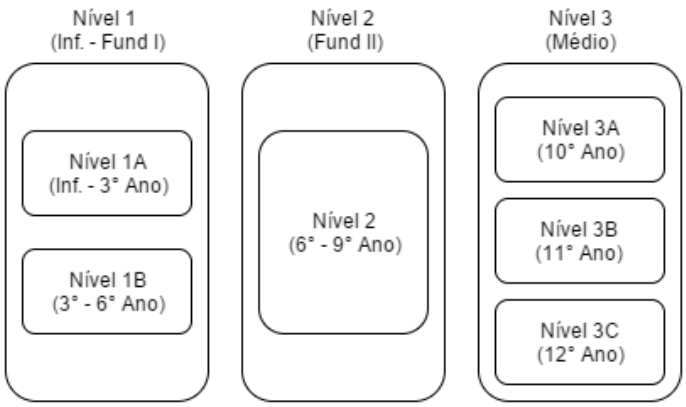


Figura 2 - Níveis CSTA (CSTA, 2011)

Focado no ensino fundamental II, o nível 2 do CSTA enfatiza a utilização, por parte dos alunos, do pensamento computacional para a resolução de problemas relevantes, não só para eles, mas para o mundo em torno deles e como suas soluções impactam a sociedade. O objetivo principal é proporcionar aos alunos a percepção de que eles são solucionadores de problemas proativos e capacitados (CSTA, 2011). Para isso pode haver uma disciplina explicita de computação ou incorporada a outras disciplinas (CSTA, 2011). Para que os alunos desenvolvam habilidades relacionadas a computação, eles devem ser estimulados a decompor problemas em problemas menores, trabalhar em equipe e desenvolver algoritmos. Por isso o *framework* de currículo do CSTA divide os seus objetivos específicos em várias áreas de conhecimento computacional como mostra a Figura 3.

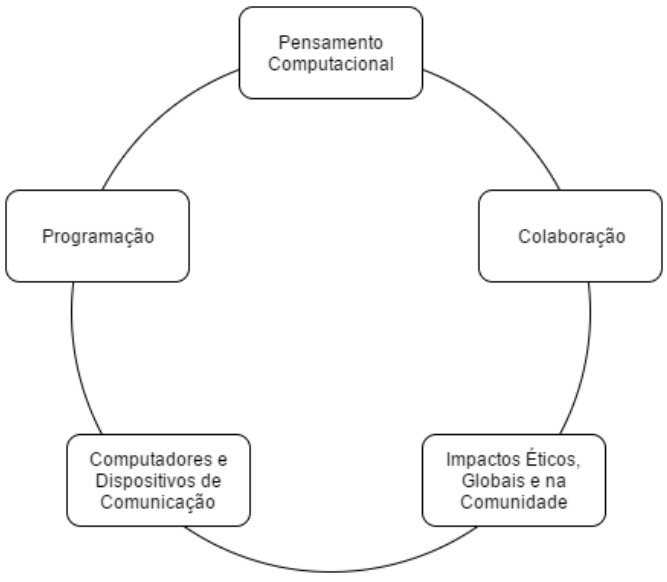


Figura 3 - Áreas do ensino de computação no Ensino Básico (CSTA, 2011)

Pensamento Computacional: é um conjunto de habilidades e competências, das quais envolvem abstração e decomposição para resolução de problemas utilizando algoritmos e recursos computacionais (WING, 2006). O pensamento computacional pode ser usado em todas as disciplinas para resolver problemas, criar novos sistemas, desenvolver novos conhecimentos, e entender as limitações da computação (CSTA, 2011).

Colaboração: a computação é uma área intrinsecamente colaborativa. Um progresso significativo na ciência da computação é, dificilmente, alcançado por uma pessoa trabalhando sozinha. Geralmente, os trabalhos da área são projetados, testados, implementados por equipes grandes, por profissionais da computação trabalhando em conjunto. Isso possibilita aos alunos entender que a colaboração no trabalho pode dar oportunidades de aprendizado através da relação com os colegas e a disciplina necessária para que o processo seja eficiente (CSTA, 2011).

Computador e Dispositivos de Comunicação: os alunos devem compreender os elementos que compõem os computadores modernos, dispositivos de comunicação e as conexões entre dispositivos. Eles devem ser capazes de entender o quanto a internet é um meio facilitador nas comunicações e de como ter um bom comportamento no seu uso. Os alunos devem usar terminologia adequada quando se comunicarem sobre tecnologia (CSTA, 2011).

Impactos Éticos, Globais e na Comunidade: é importante que os alunos saibam que os computadores e internet são elementos multiculturais e internacionais, e portanto, devem compreender os impactos sociais, positivos e negativos, que podem ser causados. Os alunos também devem entender as normas éticas de uso da internet, de conteúdos e serviços providos nela, tais como os princípios da privacidade, da segurança na internet, das licenças de software e direitos autorais (CSTA, 2011).

Prática Computacional e Programação: os alunos devem ser capazes de criar e organizar páginas da internet, usar a programação para resolver problemas, selecionar arquivos e banco de dado, usar ferramentas, APIs e bibliotecas adequadas para a resolução algorítmica dos problemas computacionais. Os alunos também devem aprender sobre a complexidade dos algoritmos, sobre o uso adequado de

elementos da linguagem, sobre técnicas de reutilização de código, sobre boas práticas de programação (CSTA, 2011).

Dentre os objetivos específicos definidos para esta área do conhecimento definidos pelo CSTA, na Tabela 1 são detalhados aqueles que tem foco neste trabalho que é voltado ao nível 2.

Tabela 1 - Objetivos de aprendizagem de Programação, nível 2 (CSTA, 2011)

	O aluno será capaz de:
1.	Selecionar ferramentas e recursos tecnológicos adequados para realizar tarefas variadas e solucionar problemas.
2.	Conceber, desenvolver, publicar e apresentar produtos (p. ex., aplicações móveis) usando recursos de tecnologia que correspondem aos conceitos do currículo.
3.	Demostrar compreensão de algoritmos e sua aplicação na prática.
4.	Implementar soluções utilizando uma linguagem de programação, incluindo: o comportamento de laços, instruções condicionais, lógica, expressões, variáveis e funções.
5.	Demonstrar receptiva disposição para resolver e programar problemas indeterminados (p. ex., conforto com complexidade, criatividade).

Com estes objetivos alcançados os alunos já terão uma noção clara e objetiva do que é a área da computação e além disso já terão competência computacional para qualquer área do conhecimento da vida profissional.

Computação no Ensino Básico é muitas vezes ensinada por meio de unidades instrucionais que ensinam a programação de código (WING, 2006). Neste contexto, pode se identificar diferentes tipos de ensino de programação. Uma forma de ensino guia o aluno a executar exercícios de programação predefinidos, como por exemplo completar um trecho de código faltante. Geralmente esses exercícios tem apenas uma resposta correta e, portanto, a análise e avaliação do mesmo refere-se apenas a uma comparação com um gabarito.

Muitas atividades de computação focam na criação de soluções para problemas do mundo real, em que as soluções são artefatos de software, como por exemplo, jogos, animações ou até mesmo apps para celulares para solucionar problemas da comunidade. Essas atividades de computação estão ligadas a

aprendizagem baseada em problemas, que são problemas complexos e abertos a várias possíveis soluções (LYE & KOH, 2014).

2.3 ANÁLISE E AVALIAÇÃO DE CÓDIGO

Uma forma de ensino e avaliação de aprendizagem de programação é por meio de trabalhos práticos. Para que o aluno adquira a competência em programação no nível de aplicação da Taxonomia de Bloom (BLOOM, 1956), se faz necessária a prática em trabalhos práticos de programação (SINTHSIRIMANA et al., 2013). Os trabalhos práticos visam praticar de forma mais ampla o conteúdo que é ensinado, como, por exemplo, o aluno desenvolvendo o seu próprio *app*. Estes trabalhos práticos de programação também podem ser utilizados pelo professor como forma de avaliar o desempenho do aluno no processo de ensino. Cada trabalho prático de programação feito pelo aluno gera um código fonte como resultado. Este código fonte pode ser analisado e avaliado para compreender o nível de aprendizagem do aluno. Esta análise e avaliação do código é, tipicamente, feita manualmente pelo próprio professor da disciplina. Porém, para que uma avaliação seja justa e eficiente é necessária competência por parte do avaliador que também requer um esforço e tempo considerável (MEDEIROS, 1974). Isso pode tornar a avaliação destes trabalhos práticos inviável ou de baixa qualidade, e consequentemente impactar negativamente no próprio processo de ensino e/ou impedir a implementação do ensino de computação nas escolas (ZEN et al.,2011).

Para analisar e avaliar o código desenvolvido pelo aluno podem ser utilizadas ferramentas de software que auxiliam o instrutor na análise e avaliação do código de forma (semi-)automática (RASHKOVITS, LAVY, 2013). Estas ferramentas de software se chamam analisadores de código ou *autograders*.

Um **Analisador de Código** é um sistema de software que analisa o código para determinar suas características, defeitos e/ou componentes utilizados no desenvolvimento do código (AHO, SETHI, ULLMAN, 1986) (WICHMANN, et al., 1995).

Analisadores de código são utilizados para diversos fins. Frequentemente são utilizados em processos de compilação de código, ou seja, no processo de tradução do código em linguagem de programação em linguagem de máquina. Neste caso a análise é feita em três etapas (AHO, SETHI, ULLMAN, 1986):

- Etapa 1: análise léxica em que o analisador lê os caracteres do código e agrupa-os em *tokens* (identificadores) para uso nas etapas posteriores;
- Etapa 2: análise sintática em que o analisador verificará se a cadeia de *tokens* gerados pelo analisador léxico pode ser gerada pela linguagem de programação, gerando uma estrutura hierárquica dos *tokens*; e
- Etapa 3: análise semântica, usando a estrutura hierárquica da análise sintática o analisador semântico identifica os operadores e operandos das expressões e enunciados e verifica se fazem sentido para linguagem.

A análise de código também pode ser usada para outras finalidades, como a análise da qualidade do código desenvolvido (YULIANTO et al., 2014). Neste caso, podendo ser dividida em dois tipos de análise: a estática e a dinâmica (YULIANTO et al., 2014). A análise estática analisa o código fonte sem a necessidade de executar a aplicação e é usada para detectar problemas no código, incluindo potenciais *bugs,* complexidade desnecessária e partes de código que necessitam manutenção. Neste caso é incluída a analise sintática, léxica e semântica.

A análise dinâmica analisa os erros gerados em tempo de execução da aplicação e ajuda a avaliar a corretude da aplicação, ou seja, verifica se a aplicação realmente resolve o problema proposto e está de acordo com sua especificação (TRUONG et al., 2004).

O foco do presente trabalho está voltado a análise estática, já que por meio dela também pode-se analisar a qualidade do código no contexto do ensino.

A **análise estática** é um método de aferição do programa desenvolvido examinando o código fonte sem a necessidade de executar o programa (ROUSE, 2006). Esse processo fornece a compreensão da estrutura do código e ajuda a verificar se o código adere as normas industriais, mundiais, de mercado ou até mesmo de ensino (ROUSE, 2006).

Os **analisadores estáticos de código** podem ajudar programadores, desenvolvedores, ou instrutores a fazer uma análise estática de código, permitindo assim que se possa encontrar erros que não se manifestam ou permitindo avaliar um determinado código (ROUSE, 2006). Analisadores estáticos são ferramentas de software que varrem o código fonte (SOMMERVILLE, 2007) e são as ferramentas mais comuns para analisar e avaliar de forma automática os trabalhos práticos de programação (STRIEWE et al., 2014), sendo o foco do presente trabalho.

Não existe um processo padrão de análise estática, porém, de forma geral um processo de análise estática de código, tipicamente, segue o processo apresentado na Figura 4 com base em Striewe (2014) e Townhidnejad (2002).

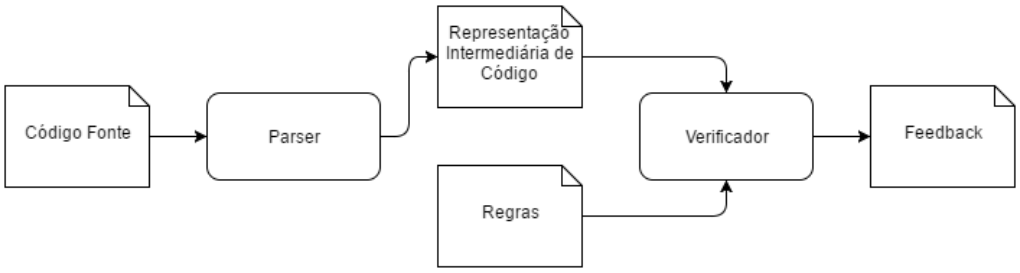


Figura 4 - Processo de análise estática (STRIEWE et al., 2014)

A entrada do analisador estático é o código fonte da aplicação desenvolvida. O primeiro passo do analisador é chamado de *parsing* que é muito parecido com as análises léxica e sintática de um compilador em que os elementos da linguagem presentes no código são identificados por *tokens e* classificados, gerando assim uma estrutura de dados (grafo ou árvore) com toda a informação necessária.

Esta estrutura de dados é utilizada pelo o que é chamado de verificador. O verificador analisa e avalia todos os elementos presentes na estrutura de dados criada pelo *parser.* O verificador tem definido os aspectos (regras de análise) a serem analisados do código fonte. As regras de análise são um conjunto de regras que informam ao verificador quais pontos e componentes do código devem ser analisados e de que forma devem ser analisados. As regras também definem parâmetros sobre os limites do que é ou não aceitável num código fonte.

Analisando diferentes propostas de implementações de analisadores estáticos de código (SINTHSIRIMANA, et al., 2013) (DR. SCRATCH, 2014), podemos observar que essas regras de análise podem ser definidas de duas formas no verificador:

- O verificador é fixo e todas as regras de análise são previamente definidas e não podem ser alterados; ou
- O verificador é configurável e então receberá as regras de análise no momento que será executado, e, portanto, serão analisados e avaliados do código fonte somente os aspectos que o avaliador desejar.

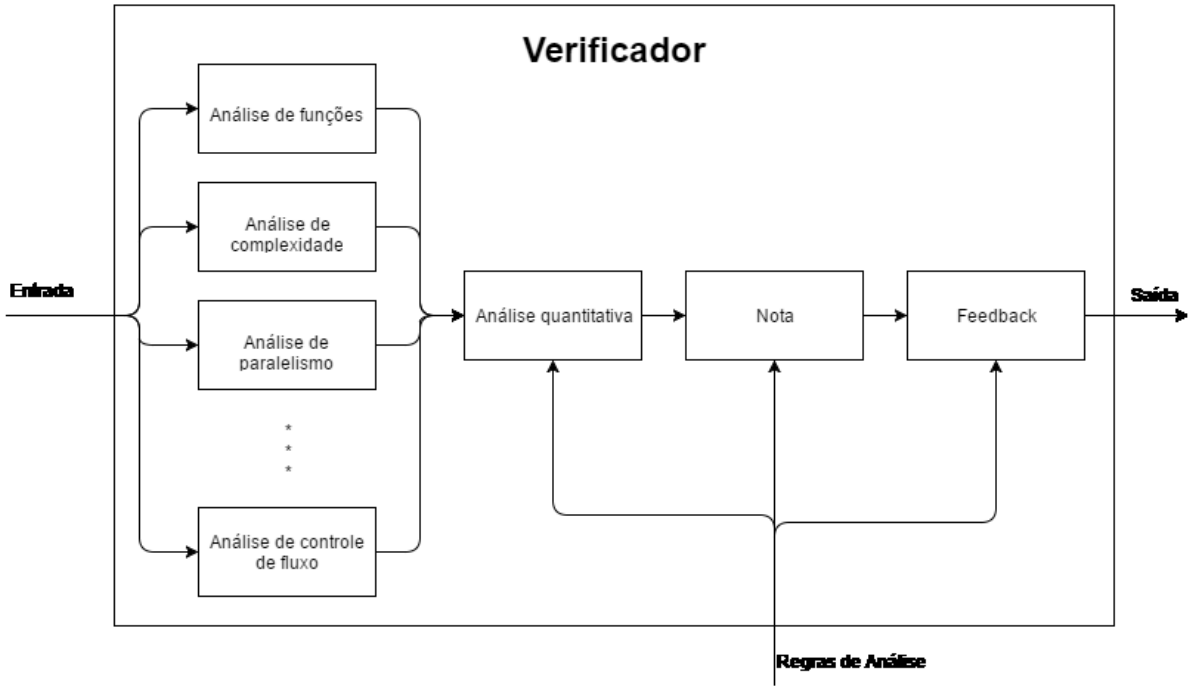


Figura 5 - Visão do fluxo do verificador adaptado voltado ao ensino (Sinthsirimana et al., 2013)

O verificador pode ser dividido em partes, cada parte fica responsável por uma tarefa ou um pequeno conjunto de tarefas. O verificador recebe como entrada a estrutura de dados gerada pelo *parser* que é o código fonte estruturado:

1º Estágio: o verificador fará análises de aspectos pontuais do código por meio de algoritmos implementados capazes de analisar funções declaradas no código, complexidade, paralelismo, entre outros.

2º Estágio: é reunida toda a informação obtida em cada aspecto analisado no estágio anterior. Com base nas regras de análise determina-se quais aspectos serão realmente considerados na análise do código como um todo e o quanto cada aspecto é significativo para a conclusão da análise.

3º Estágio: é atribuída uma nota ao código de acordo com as informações obtidas nas etapas anteriores, avaliando assim o seu desempenho.

4º Estágio: é criado uma base de informação sobre o código analisado verificando se o mesmo cumpre ou não as determinações das regras de análise que gera um feedback ao avaliador sobre os aspectos positivos e negativos do código. Além disso, esse *feedback* pode incluir dicas sobre o que pode ser melhorado no código analisado.

Cada linguagem de programação tem suas peculiaridades, portanto, geralmente cada linguagem demanda de um analisador específico. Hoje existem analisadores de código para as diversas linguagens de programação, como p.ex. PHP (SINTHSIRIMANA et al., 2013), JAVA (TRUONG et al., 2004), C e C++ (VIEGA, JOHN et al., 2000).

2.3.1 ANÁLISE DE CÓDIGO APLICADA À EDUCAÇÃO

O objetivo da análise de código aplicada a educação é analisar e avaliar o código de um trabalho prático desenvolvido pelo aluno no processo de ensino de computação.

Não há uma definição única sobre quais aspectos devem ser analisados pelo verificador e de modo geral pelo analisador estático. Porém, como em geral a avaliação de trabalhos de alunos foca na avaliação do grau de atendimento dos objetivos de aprendizagem. Assim, estes aspectos a serem avaliados devem ser derivados dos objetivos.

Existem também estudos que perceberam alguns hábitos de programação em estudantes que são contrários aos hábitos aceitáveis na computação (MORENO & ROBLES, 2015). Voltado ao foco do presente trabalho os aspectos importantes que, tipicamente, devem ser analisados neste contexto são (RASHKOVITS & LAVY, 2013) (MORENO & ROBLES, 2015):

- **Abstração e Modularização:** O código deve ser eficientemente organizado em classes e as classes devem ser organizadas em hierarquias. Cada classe representa um único conceito e tem todos os atributos e métodos necessários.
- **Projeto de métodos:** Um método deve ser relativamente pequeno e deve executar uma única tarefa ou um conjunto altamente relacionado de pequenas tarefas.
- **Legibilidade de código:** O nome de classes, variáveis e métodos deve ser significativo para facilitar o entendimento do código.
- **Paralelismo:** O código deve priorizar o paralelismo na execução de atividades não relacionadas ou que não dependem uma da outra para execução.

- **Sincronização:** O código deve conter componentes capazes de sincronizar a execução de vários elementos dinâmicos da aplicação.
- **Controle de fluxo:** As instruções, expressões e chamadas de métodos devem ser corretamente ordenadas. O uso de condicionais, laços e chamada de sub-rotinas deve ser eficiente.
- **Interatividade com usuário:** A aplicação deve ser interativa com o usuário. O usuário deve ser participante.
- **Representação de dados:** A aplicação deve representar e armazenar os dados coletados de forma eficiente, e de forma a garantir a consistência dos dados.

Além desses aspectos, em aplicativos para dispositivos moveis, pode-se incluir a análise de alguns pontos específicos que são, geralmente, encontrados em apenas aplicações deste gênero. Estes aspectos incluem (SHERMAN, 2015):

- Compartilhamento de dados;
- Uso de serviços web públicos;
- Uso de acelerômetro e sensores de orientação;
- Reconhecimento de localização.

No foco do presente trabalho necessita-se então da medição destes aspectos com base em trabalhos práticos, que possuam várias possíveis soluções a serem desenvolvidas pelos alunos. Neste caso a análise e avaliação de código requer mais flexibilidade, pois, possivelmente existem diversas soluções corretas ao problema. Assim, se faz necessário um analisador de código que possa fazer uma análise mais flexível e livre do código.

Neste contexto, a análise e avaliação do programa baseia-se no pressuposto de que certos atributos mensuráveis podem ser extraídos do programa, avaliando se os alunos aprenderam o que era esperado utilizando uma rubrica. A rubrica usa medidas descritivas para separar os níveis de desempenho em uma determinada tarefa, delineando os vários critérios associados às atividades de aprendizado (WHITTAKER et al., 2001). Por meio da rubrica a pontuação de cada critério é definida, permitindo um *feedback* instrucional.

O *feedback* instrucional, resultado da avaliação, tipicamente inclui uma nota e um *feedback* ao aluno. No Brasil não existe um consenso e nem regra de como as notas devem ser dadas às provas e trabalhos práticos nas escolas. Cada estado ou município fica encarregado de definir qual padrão seguir, e mesmo assim podem haver diferenças entre a rede estadual e municipal além de pública e privada. Como exemplo na cidade de Florianópolis - SC a RESOLUÇÃO CME Nº02/2011 Art. 7º diz que as notas podem ser em "[...]parecer descritivo que revele o diagnóstico do processo de aprendizagem" e em número de 1 a 10.

O *feedback* é importante no contexto educacional para facilitar aprendizagem (MORENO, 2004). O *feedback* é uma técnica para orientar os alunos a pensarem sobre suas respostas e fornecer informações para direcionar a mudança de sua forma de pensar e agir (BILAL et al., 2012). Pode ser considerado uma resposta as ações do aluno, indicando a exatidão das respostas ou sendo mais amplo provendo dicas, sugestões e exemplos relacionados ao conteúdo (BLACK & WILIAN, 1998). Portanto é importante que ao final da avaliação sejam dados uma nota e um *feedback* ao aluno.

2.3 APP INVENTOR

Existem muitas linguagens de programação hoje em dia, cada uma com suas peculiaridades e objetivos (IEEE, 2016). A maior parte das linguagens de programação hoje utilizadas no desenvolvimento profissional são linguagens de programação textual. Porém, com o objetivo de engajar crianças e jovens na área da computação e para facilitar a aprendizagem de computação nesta faixa etária, são indicados o uso de linguagens de programação visuais (RESNICK et al., 2009).

Linguagens de programação visuais baseado em blocos permitem criar sistemas de software ou aplicativos arrastando e encaixando blocos. Nas linguagens visuais todos os componentes das linguagens já estão definidos, normalmente, em forma de figuras ou blocos, o usuário deverá colocar esses blocos numa ordem que faça sentido ao programa e então a aplicação começa a ser programada. Desta maneira ele não precisará escrever linhas de código textuais, eliminando a necessidade de se entender e memorizar as complexas sintaxes de linguagens textuais.

Umas das linguagens de programação visual é o Blockly (BLOCKLY, 2017). Criado pela Google, tem como objetivo ser uma biblioteca que sirva como base para o desenvolvimento de ambientes de programação.

O **App Inventor** (APP INVENTOR, 2017) é um ambiente de programação visual criado pela Google e atualmente mantido pelo Instituto de Tecnologia de Massachusetts (MIT). O App Inventor utiliza a linguagem de programação visual Blockly para permitir a criação de aplicativos para *smartphones e tablets* com sistema operacional Android. O App Inventor, por meio de uma interface gráfica simples, permite que pessoas inexperientes em programação ou até mesmo crianças tenham a habilidade de criar do mais básico ao mais completo aplicativo em uma hora ou menos (APP INVENTOR, 2017). O objetivo da criação do App Inventor foi democratizar o desenvolvimento de software dando apoio a todas as pessoas, principalmente aos jovens, na transição de simples consumidores de tecnologia para criadores de tecnologia (APP INVENTOR, 2017). Em 2016 o App Inventor já era utilizado por mais de 5 milhões de usuários em mais de 195 países (APP INVENTOR, 2017).

O App Inventor foi implementado na linguagem de programação Java e o seu código fonte é aberto (APP INVENTOR, 2017). Ele é um grande sistema dividido em múltiplos projetos, cada um deles depende de diferentes tecnologias de código aberto.

O App Inventor é executado paralelamente e em sincronia em dois módulos. No lado do usuário executando no navegador de internet, é onde a aplicação é desenvolvida pelo programador e outro no lado do servidor, onde a aplicação desenvolvida é processada e salva.

O App Inventor permite que o projeto de app criado seja exportado em um arquivo compactado no formato *aia*. Na versão *nb162a* do App Inventor, este arquivo inclui todas as informações do app, como imagens e sons utilizados e os códigos produzidos pelos blocos da lógica e componentes visuais presentes no app. Os códigos referentes a parte visual do app ficam registrados em arquivos no formato *scm* que por sua vez encapsulam uma estrutura *json* que contém todos os componentes utilizados. Os códigos produzidos pela parte lógica do app ficam registrados em arquivos no formato *bky* que por sua vez encapsulam estruturas *xml* com toda a lógica implementada.

Criando uma Aplicação

O desenvolvimento de aplicações no App Inventor é dividido em duas grandes áreas: Designer e Blocos.

A área **Designer** é utilizada para a programação dos componentes referentes a interação da aplicação com o usuário e de modo geral a conectividade da aplicação com outros componentes exteriores.

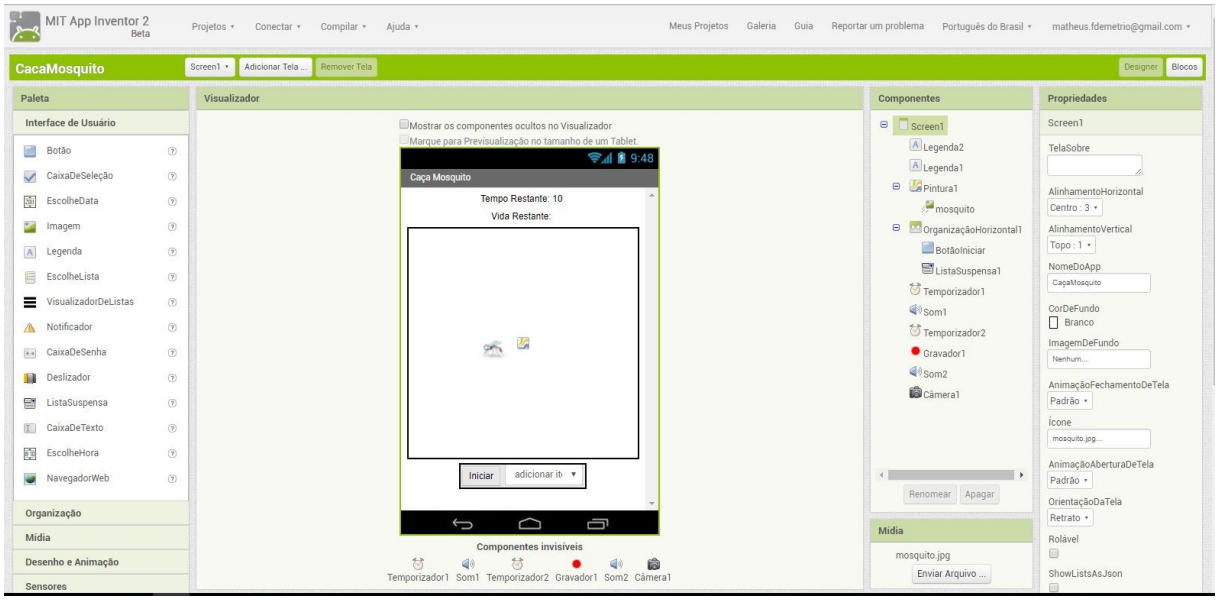


Figura 6 - Tela Designer do App Inventor (APP INVENTOR, 2017)

Nesta área é possível programar componentes responsáveis por essa interação como temporizadores, sons e a parte visual. Com isso é possível criar caixas de texto, botões, alterar cores entre outras possibilidades. Na Tabela 2 são apresentados os componentes da área **Designer** do App Inventor, bem como uma breve descrição de cada componente.

Tabela 2 - Componentes do Designer

Grupo	Componentes	Descrição
Interface de Usuário	Botão; Caixa de Seleção; Escolhe Data; Imagem; Legenda; Escolhe Lista; Visualizador de Listas; Notificador; Caixa de Senha; Deslizador; Lista Suspensa; Caixa de Texto; Escolhe Hora; Navegador Web.	Criação da parte visual do app. Todos os elementos visíveis da aplicação ficam neste grupo.
Organização	Organização Horizontal; Horizontal Scroll Arrangement; Organização em	Auxilia na organização dos elementos visíveis do grupo de interface com usuário.

	Tabela; Organização Vertical; Vertical Scroll Arrangement.	
Mídia	Câmera de Vídeo; Câmera; Escolhe Imagem; Tocador; Som; Gravador; Reconhecedor de Voz; Texto para Falar; Reprodutor de Vídeo; Tradutor Yandex.	Todos os componentes de mídia que podem ser usados no desenvolvimento da aplicação.
Desenho e Animação	Bola; Pintura; Sprite Imagem.	Componentes que permitem ao usuário desenhar e visualizar animações.
Sensores	Sensor acelerômetro; Codigo de barras; temporizador; Gyroscope; Localização; Near Fiels; Orientação; Pedometro; Proximidade.	Componentes que obtém informações dos sensores presentes no dispositivo em que o *app* será instalado.
Social	Escolher contato; escolhe email; Ligação; Escolhe Número telefone; Compartilhamento; SMS; Twitter.	Componentes que permitem a comunicação da aplicação com outros aplicativos sociais.
Armazenamento	Arquivo; Fusiontables; TinyDB; TinyWebDB.	Componentes que permitem a criação de bancos de dados para armazenar dados.
Conectividade	Iniciador de Atividade; Cliente Bluetooth; Servidor Bluetooth; Web.	Componentes que permitem a conectividade da aplicação com outros dispositivos.

A área de **Blocos** é utilizada para a programação da lógica, é a parte não visível da aplicação.

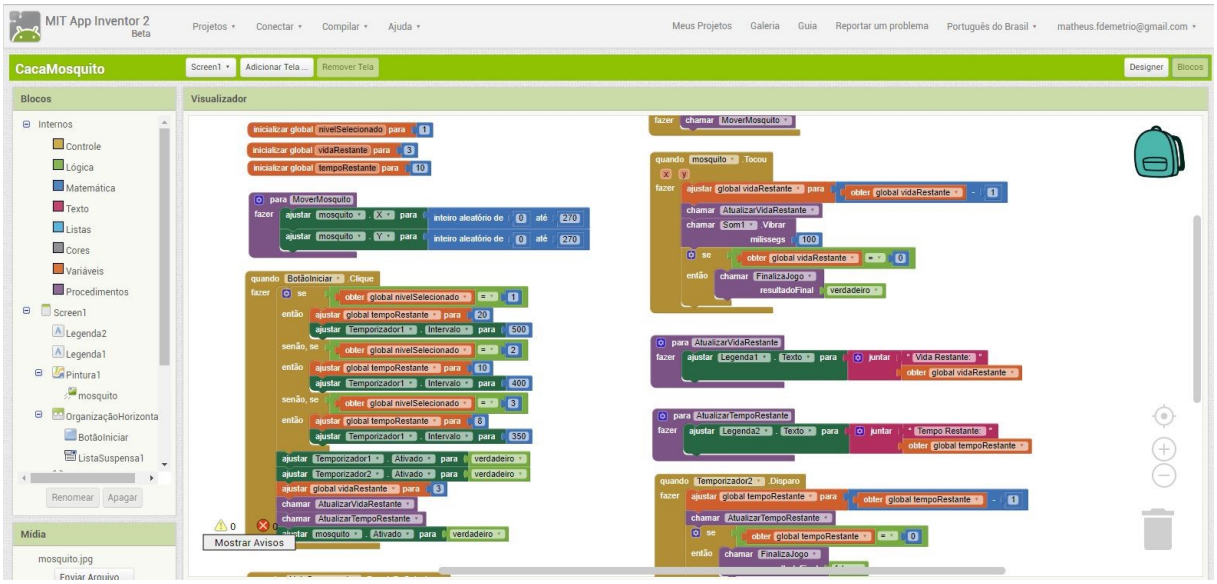


Figura 7 - Tela Blocos do App Inventor (APP INVENTOR, 2017)

Esta parte é a responsável por controlar as ações da interface de usuário, nela são processadas todas as informações obtidas por meio da interface de usuário. Aqui existem diversos blocos tais como: blocos de controle, com os quais é possível criar funções de controle ou funções condicionais entre outras; blocos de matemática que

permitem criar expressões e funções matemáticas; blocos de listas usados para manipular listas de dados; e algumas outras opções. Todos os blocos são apresentados e detalhados nas Tabelas 3 a 10.

Controle: Comandos responsáveis pelas funções de controle da execução da aplicação, nele estão disponíveis blocos importantes como laços, condicionais e iterações em listas.

Tabela 3 - Blocos de Controle

Bloco	Descrição
If e if else	Testa uma condição dada. Se a condição for verdadeira, executa uma sequência de blocos, senão, ignora esta sequência de blocos.
For each – from – to	Executa os blocos na seção para cada valor numérico no intervalo - **a partir de** – e - **terminando em** -, incrementando o número a cada vez executado.
For each list item	Executa uma sequência de blocos para cada item de uma lista.
While	Testa uma condição repetidas vezes, enquanto for verdadeira executa uma sequência de blocos.
If then else	Testa uma condição, se for verdadeira, executa os blocos do then, se for falsa, executa os blocos do else.
Do	Executa um bloco e retorna um resultado.
Evaluate but ignore result	Usado como bloco fictício, o bloco em que este esteja conectado será executado, mas seu retorno será ignorado.
Open another screen	Abre uma tela com o nome dado.
Open another screen with start value	Abre uma tela com o nome e um valor passado.
Get start value	Retorna o valor atual dado a uma tela.
Close screen	Fecha a tela atual.
Close screen with value	Fecha a tela atual e retorna o valor passado a ela.
Close application	Fecha a aplicação.
Get plain start text	Retorna o texto que foi passado a tela quando foi iniciada por outro *app*.
Close screen with plain text	Fecha a tela e passa o texto ao *app* que iniciou ela.

Lógica: Componentes responsáveis pelas operações lógicas sobre as variáveis.

Tabela 4 - Blocos Lógicos

Bloco	Descrição
True, false	Representa a constante com valor verdadeiro ou falso.
Not	Negação lógica.

=, ≠	Testa se os argumentos são iguais ou diferentes.
and	Testa se todas as condições lógicas são verdadeiras.
or	Testa se alguma condição lógica é verdadeira.

Matemática: Componentes responsáveis por operações matemáticas sobre variáveis. Há operações básicas como soma, subtração, multiplicação e divisão, e operações mais complexas como as trigonométricas.

Tabela 5 - Blocos Matemáticos

Bloco	Descrição
Basic number block	Representa qualquer número positivo ou negativo.
=, ≠, >, ≥, <, ≤	Blocos com as operações de igualdade ou desigualdade.
Min, max	Retorna o maior ou menor valor.
Sqrt, abs, -, log, e^, round, ceiling, floor	Operações unárias padrões.
Modulo of, remainder of, quotient if	Operações de módulo, resto e quociente.
Sin, cos, tan, asin, acos, atan	Operações trigonométricas.
Convert radians to degrees, convert degrees to radians	Conversão de graus em radianos e radianos em graus.
+, -, *, /	Operações de soma, subtração, multiplicação e divisão.
Integer random	Retorna número inteiro entre valores passados.
Random fraction	Retorna uma fração randômica.
Random set seed to	Define a semente para a geração de números randômicos.
Format as decimal	Formata um número decimal em uma quantidade de casas decimais definidas.
Is a number?	Retorna verdadeiro se o objeto passado é um número.
Convert number	Converte um número em binário, hexadecimal ou decimal.

Texto: Componentes responsáveis pela manipulação de textos. Estão incluídas operações de junção de texto, verificação de partes de um texto, divisão de texto, entre outros.

Tabela 6 - Blocos de texto

Bloco	Descrição
String “”	Contém um texto (*string*)
Join	Concatena *strings*.
length	Retorna quantidade de caracteres da *string*.
Is empty	Retorna verdadeiro se a *string* não contém caracteres.
compare texts < > =	Compara *strings*, se é “maior” ou “menor” alfabeticamente, ou *string* igual.
Trim	Remove qualquer espaço que exista na *string* e retorna.
Upcase	Retorna a *string* passada formatada em letras maiúsculas.
Downcase	Retorna a *string* passada formatada em letras minúsculas.

Starts at	Retorna a posição em que começa a *string* passada em um texto.
Contais	Retorna *true* se uma parte aparece no texto.
Split at first	Divide o texto em duas partes usando a localização da primeira ocorrência do ponto de divisão.
Split at first of any	Divide o texto em dois itens de lista.
Segment	Retorna o trecho do texto de início e tamanho passados por parâmetro.
Replace all	Retorna um novo texto em que foram substituídas todas as ocorrências de determinada *substring*.

Lista: Componentes responsáveis pela criação e manipulação de listas de dados. Muito utilizado em aplicações em que o volume de informações é grande e se torna inviável armazenar em variáveis.

Tabela 7 - Blocos de Lista

Bloco	Descrição
Create empty list	Cria uma lista vazia.
Make a list	Cria uma lista a partir dos blocos definidos, se não definir blocos, cria uma lista vazia.
Add items to list	Adiciona itens ao final da lista.
Is in list?	Retorna verdadeiro se o elemento está na lista.
Length of list	Retorna a quantidade de itens na lista.
Is list empty?	Retorna verdadeiro se a lista está vazia.
Pick a random item	Obtém um item qualquer da lista.
Index in list	Retorna a posição que determinado item está na lista.
Select list item	Seleciona o item que está em determinada posição na lista.
Insert list item	Insere item na lista na posição informada.
Replace list item	Substitui item da lista por novo na posição informada.
Remove list item	Remove item da lista na posição informada.
Append to list	Adiciona itens da segunda lista no final da primeira.
Copy list	Faz uma cópia da lista incluindo sublistas.
Is a list?	Retorna verdadeiro se o objeto é uma lista.
List to csv row	Interpreta a lista como uma linha e retorna um arquivo CSV.
List from csv row	Captura um texto em CSV formatado em uma linha e cria uma lista.
List to csv table	Interpreta a lista como uma tabela e retorna um arquivo CSV.
List from csv table	Captura um texto em CSV formatado em uma tabela e cria uma lista.
Lookup in pairs	Usado para pesquisar informações em uma estrutura em pares parecida com um dicionário.

Cores: Componentes responsáveis por determinar as cores do aplicativo, podendo ser utilizadas cores padrões ou cores personalizadas pelo desenvolvedor.

Tabela 8 - Blocos de Cores

Bloco	Descrição
Basic color blocks	Bloco de cores básicas.
Make color	Recebe uma lista de 3 ou 4 valores que representam o valor RGB ou RGBA da cor.

Split color	Retorna uma lista com os valores do RGB.

Variáveis: Componentes responsáveis pela criação e manipulação das variáveis da aplicação. As variáveis podem ser definas globalmente ou localmente.

Tabela 9 - Blocos de Variáveis

Bloco	Descrição
Initialize global name to	Usado para criar uma variável global definindo um nome.
Get	Uma forma de obter qualquer variável que tenha sido criada no escopo atual.
Set to	Uma forma de atribuir um valor a uma variável criada.
Initialize Local name to - in (do)	Permite a criação de uma variável que será usada apenas no *DO* de um método.
Initialize Local name to - in (return)	Permite a criação de uma variável que será usada apenas no *RETURN* de um método.

Procedimentos: Componentes responsáveis pelos procedimentos da aplicação. Cada procedimento é um grupo de blocos que tem um nome e uma função bem definida. Os procedimentos também são chamados na computação de métodos ou funções.

Tabela 10 - Blocos de Procedimentos

Bloco	Descrição
Procedure do	Agrupa uma sequência de blocos e não retorna uma resposta ao final da sua execução.
Procedure result	Agrupa uma sequência de blocos e retorna uma resposta ao final da sua execução.

A análise estática de código é feita sobre a lógica da aplicação, e toda a parte lógica das aplicações desenvolvidas com o App Inventor são criadas na área de **Blocos**. Isto torna estes componentes o ponto central de análise do presente trabalho.

3. ESTADO DA ARTE

Neste capitulo é apresentado o estado da arte. O estado da arte é levantado por meio de estudo de mapeamento com o objetivo de encontrar e analisar as ferramentas de análise estática de código de linguagens de programação visuais para o ensino de computação.

3.1 DEFINIÇÃO DO PROTOCOLO DE REVISÃO

Questão de Pesquisa. Esse estudo tem como questão central de pesquisa a seguinte pergunta: “Quais analisadores estáticos de código existem para análise de código de linguagens de programação visuais no contexto de ensino de programação no Ensino Fundamental II?”

Termos de Busca. De acordo com a pergunta de pesquisa são definidos os termos de busca. Foram realizadas buscas informais com o objetivo de calibrar a *string* de busca e obter a maior quantidade de artigos que satisfaçam os critérios de inclusão. Os termos de busca derivados são apresentados na Tabela 11, incluindo sinônimos e traduções dos termos.

Tabela 11 - Termos de Busca

Termo	**Sinônimos**	**Tradução para inglês**
Analisador estático de código	Analisador de código, Analisador de programação de computador.	Static code analyzer, Code analyzer, Computer programming analyzer
Linguagem de programação visual	App Inventor, Scratch, Blockly, Snap!, App Lab.	Visual programming language, block-based languages
Ensino	Educação, Aprendizagem.	Education, teaching, learning.

***String* de busca genérica**

(“Static code analyzer” OR “code analyzer” OR “Computer programming analyzer”) AND (“visual programming language” OR “block-based languages” OR “App Inventor” OR “Scratch” OR “Blockly” OR “Snap!” OR “App Lab”) AND (“Education” OR “teaching” OR “learning”)

Bases de pesquisa. Para uma ampla busca nas bases de dados digitais na área do presente trabalho, são buscados artigos científicos publicados entre os anos de 2010 (data do lançamento da primeira versão do App Inventor) e 2016 em Português e Inglês. Por meio do portal CAPES[2] as fontes para a realização da busca são:

- IEEE Xplore[3]
- ACM Digital Library[4]
- ScienceDirect[5]
- Springer[6]
- Wiley[7]

Também é utilizada a ferramenta de busca da Google para acadêmicos, o Google Scholar[8], bem como o site do App Inventor e seu fórum e o fórum do App Inventor[9] para educadores onde é possível encontrar alguns trabalhos desenvolvidos pela comunidade.

De acordo com a sintaxe de cada uma desta base são definidas as *strings* de busca de cada base.

Tabela 12 - Strings de Busca por Base

IEEE Xplore	*(“Static code analyzer” OR “code analyzer” OR “Computer programming analyzer”) AND (“visual programming language” OR “*block-based languages*” OR “App Inventor” OR “Scratch” OR “Blockly” OR “Snap!” OR “App Lab”) AND (“Education” OR “teaching” OR “learning”)*

[2] Portal desenvolvido pelo Ministério da Educação, onde é possível encontrar artigos científicos de relevância - https://www.capes.gov.br/
[3] http://ieeexplore.ieee.org/
[4] http://portal.acm.org
[5] http://www.sciencedirect.com
[6] http://link.springer.com/
[7] http://www.wiley.com/
[8] https://scholar.google.com.br/
[9] http://teach.appinventor.mit.edu/

ACM Digital Library	*(“code analyzer”) AND (“visual programming language” OR “*block-based languages*” OR “App Inventor” OR “Scratch” OR “Blockly” OR “Snap!” OR “App Lab”) AND (“learning”)*
ScienceDirect	*(“Static code analyzer” OR “code analyzer” OR “Computer programming analyzer”) AND (“visual programming language” OR “block-based languages” OR “App Inventor” OR “Scratch” OR “Blockly” OR “Snap!” OR “App Lab”) AND (“Education” OR “teaching” OR “learning”)*
Springer	*(“code analyzer”) AND (“visual programming language” OR “block-based languages” OR “App Inventor” OR “Scratch” OR “Blockly” OR “Snap!” OR “App Lab”) AND (“education”)*
Wiley	*(“code analyzer”) AND (“visual programming language” OR “block-based languages” OR “App Inventor” OR “Scratch” OR “Blockly” OR “Snap!” OR “App Lab”) AND (“learning”)*
Google Scholar	*(“code analyzer”) AND (“visual programming language” OR “scratch”) AND (“learning”)*
App Inventor	*(“code analyzer”)*

Critérios de inclusão e exclusão. Para que a questão de pesquisa seja respondida foram definidos critérios de inclusão e exclusão para que trabalhos existentes sejam considerados ou não. Só são considerados artigos sobre analisadores estáticos de código, e que sejam relacionados a linguagens de programação visual e focados na análise no contexto do ensino de programação. São desconsiderados artigos duplicados, sobre analisadores dinâmicos, ou relacionados a programação textual ou focados em outras áreas que não o ensino, além de que serão desconsiderados analisadores para exercícios de programação fixos pré-definidos.

Critério de qualidade. Os artigos devem fornecer um grau de informação suficiente, portanto não podem ser resumidos, devem abordar com detalhes o analisador estático de código em questão e os processos de análise de código.

3.2 EXECUÇÃO DA BUSCA

A execução da busca foi realizada no mês de novembro de 2016 pelo autor do presente trabalho. A base de pesquisa que mais retornou resultados foi a ACM Digital Library, que por meio da *string* de busca retornou cerca de 100 mil resultados, tornando-se inviável a análise de todos os resultados e, portanto, a análise focou somente aos 100 primeiros itens retornados pelo mecanismo de busca. Primeiramente, foi feita uma análise dos títulos e descrições dos artigos verificando se estavam de acordo com os critérios de inclusão. Como resultado foram

identificados cinco artigos potencialmente relevantes. Muitos dos artigos não incluídos eram voltados a analisadores para atividades de programação especificas, fixas e pré-definidas como por exemplo o Tynker, e consequentemente não foram levados em consideração por não estarem alinhados ao critério de qualidade definido.

Vale ressaltar também que foi encontrado na base ACM um resumo de um artigo (*abstract*) no qual o autor informa que está desenvolvendo um analisador de código para Snap! (BALL & GARCIA, 2016). Porém, como no momento da busca não foi encontrado um artigo completo ou mais informações sobre o andamento deste projeto, o trabalho não foi incluído nesta revisão.

Após essa análise inicial, os artigos relevantes foram lidos na íntegra. Ao final foram considerados apenas as ferramentas de análise estática de código para linguagens de programação visuais no contexto de ensino que estivessem completas e prontas para o uso de instrutores. Na Tabela 13 encontram-se os resultados da busca.

Tabela 13 - Resultado da Execução da Busca

Base de pesquisa	Quantidade de resultados da busca	Quantidade dos resultados analisados	Quantidade de artigos potencialmente relevantes depois da análise dos resultados da busca	Quantidade dos artigos relevantes
IEEE Xplore	6	6	2	1
ACM Digital Library	105.000	100	9	2
ScienceDirect	221	100	0	0
Springer	140	100	0	0
Wiley	0	0	0	0
App Inventor	0	0	0	0
Google Scholar	22.300	100	11	1
Total	**127.667**	**406**	**22**	**4**

Ao final, os 4 artigos relevantes encontrados apresentaram 2 ferramentas de análise estática de código de linguagens de programação visuais no contexto de ensino conforme apresentado na Tabela 14.

Tabela 14 - Analisadores de Código Relevantes Encontrados

Dr. Scratch	• MORENO-LEÓN, J.; ROBLES, G.; ROMÁN-GONZÁLEZ, M. Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. RED. Revista de Educación a Distancia, v. 15, n. 46, 2015. • MORENO-LEÓN, J.; ROBLES, G. Analyze your Scratch projects with Dr. Scratch and assess your computational thinking skills. In: Scratch Conference. 2015. Amsterdam, Netherlands. • MORENO-LEÓN, Jesús; ROBLES, Gregorio. Dr. Scratch: A web tool to automatically evaluate Scratch projects. In: Proceedings of the workshop in primary and secondary computing education. ACM, 2015.
Ninja Code Village	• OTA, G.; MORIMOTO, Y.; KATO, H. Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Chiba/Japão 2016.

3.3 ANALISE DOS TRABALHOS ENCONTRADOS

Nesta seção são extraídas informações referentes aos analisadores estáticos de código de linguagens de programação visuais encontrados na execução da busca. Cada subseção a seguir detalha as informações das duas ferramentas encontradas.

3.3.1 DR. SCRATCH

O analisador Dr. Scratch (MORENO, 2014) é uma ferramenta Web de código fonte aberto desenvolvido na Universidad Rey Juan Carlos na Espanha. O Dr. Scratch tem como objetivo permitir que alunos e instrutores analisem seus projetos desenvolvidos na linguagem de programação visual Scratch. Além disso, o Dr. Scratch tem o objetivo de dar suporte aos professores na correção de trabalhos práticos e encorajar os alunos a melhorarem suas competências em programação.

Disponível publicamente online, para analisar um projeto criado no Scratch pode-se enviar um arquivo .sb ou .sb2 (formato do Scratch), ou simplesmente, enviar a URL do projeto ao site do Dr. Scratch, como mostrado na Figura 8.

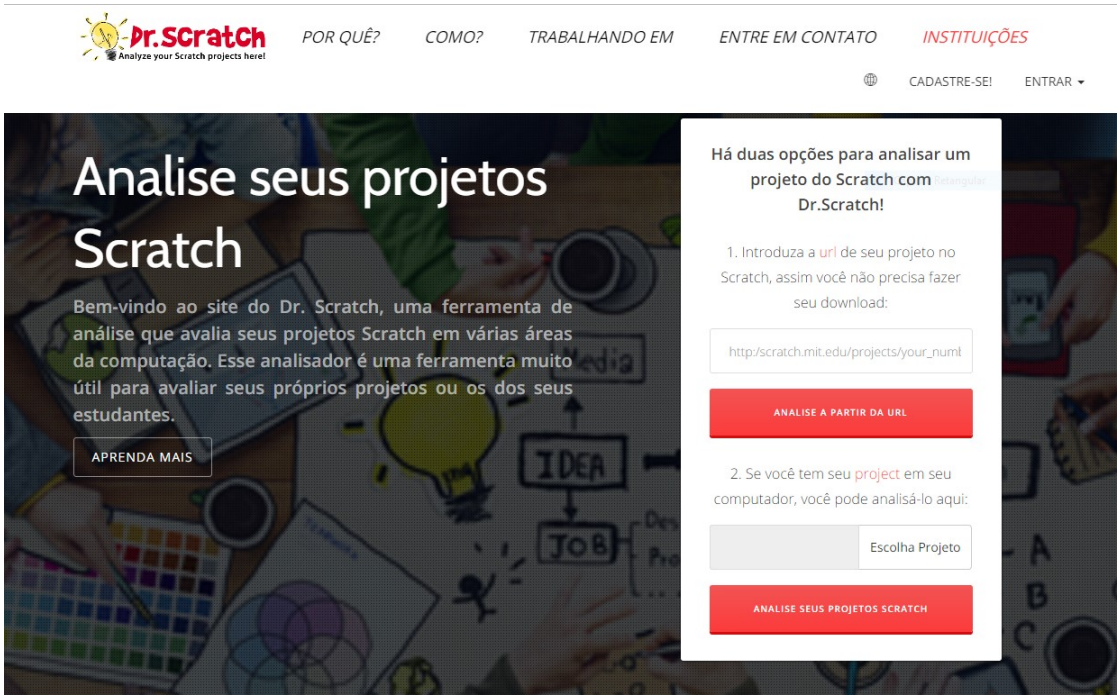


Figura 8 - Tela Inicial do Dr. Scratch

Após terminada a análise é apresentada uma tela com informações sobre os conceitos analisados bem como notas e um feedback, (Figura 9).

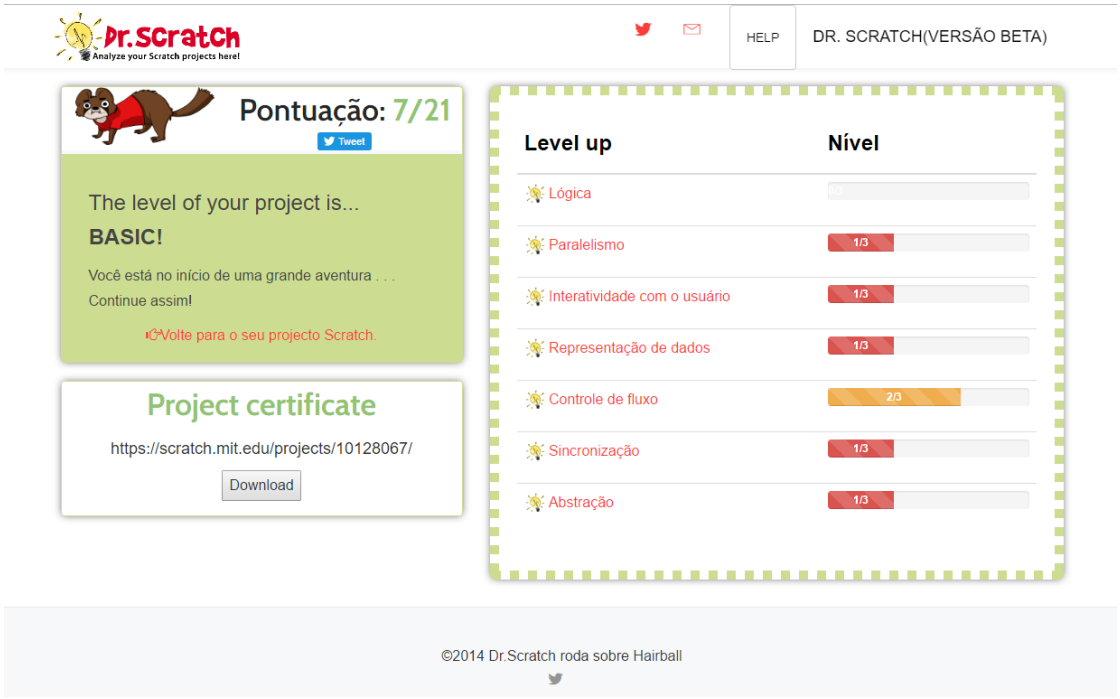


Figura 9 - Tela de Resultado de Análise do Dr. Scratch

Na primeira parte da análise que o Dr. Scratch executa, busca-se gerar estatísticas dos componentes da linguagem que são utilizados no código. Para isso é usada uma API chamada *HairBall* (HAIRBALL, 2017) que extrai todos os dados do código fonte que o usuário criou no Scratch e gera estatísticas do código.

A partir disto, é feita a analise os projetos em relação aos conceitos de programação definidos no Dr. Scratch:

- Abstração e decomposição de problemas;
- Paralelismo;
- Pensamento lógico;
- Sincronização;
- Controle de fluxo;
- Interatividade com usuário;
- Representação de dados.

Após feita a análise é atribuída uma nota de zero a três a cada conceito de programação analisado como apresentado na Tabela 15. A soma das notas dos conceitos gera a nota final que é numa escala de 0 a 21.

Tabela 15 - Rubrica Dr. Scratch (DR. SCRATCH, 2016)

Critério	0 pontos	1 ponto	2 pontos	3 pontos
Lógica	Não utilizou nenhum comando de lógica.	Utilização do comando “Se, então”.	Utilização do comando “Se, então; senão”.	Utilização de comandos de operações lógicas que combinam as condições.
Paralelismo	Não utilizou nenhum comando de paralelismo.	2 scripts iniciando com “bandeira verde”.	2 scripts com o comando “quando a tecla for pressionada” utilizando a mesma tecla ou 2 scripts com o comando “quando este ator for clicado” utilizando o mesmo ator.	2 scripts de recebimento de mensagens ou criação de clones ou 2 scripts de sensores ou 2 scripts de mudança de pano de fundo.
Interatividade com o usuário	Não utilizou nenhum comando de interatividade com o usuário.	Utilização do comando da “bandeira verde”.	Utilização de comandos de teclas/atores pressionadas, perguntas ao usuário, e botão do mouse.	Utilização de comandos de sensor de câmera e vídeo.

Representação de dados	Não utilizou nenhum comando de representação de dados.	Modificação de propriedades dos atores como coordenadas, tamanho e aparência.	Operações com variáveis.	Operações com listas.
Controle de fluxo	Não utilizou nenhum comando de controle de fluxo.	Programação de uma sequência de blocos.	Utilização dos comandos “repita x vezes” e “sempre”.	Utilização do comando “repita até que”.
Sincronização	Não utilizou nenhum comando de sincronização.	Utilização do comando “espere”.	Utilização de comandos de envio/recebimento de mensagens.	Utilização de comandos de “espere até” ou “quando o fundo muda para”.
Abstração	Não utilizou nenhum comando de abstração.	Programação de mais de 1 *script*.	Utilização e programação de funções por meio do comando “defina”.	Utilização de clones.

O Dr. Scratch é previsto a ser utilizado em dois modos: o modo individual ou modo de turma. No modo individual o usuário analisa o seu código de forma independente. Já no modo de turmas (em desenvolvimento) pode ser feita a análise de várias aplicações de forma conjunta, facilitando ao instrutor a gerencia de uma turma de alunos, por exemplo. Com uma conta de instrutor também é previsto que seja possível verificar os projetos analisados e obter estatísticas desses projetos, bem como a evolução dos alunos nas notas obtidas.

Além da funcionalidade para o instrutor, o Dr. Scratch permite ao aluno a impressão de um certificado e/ou o compartilhamento do resultado no Facebook.

3.3.2 NINJA CODE VILLAGE FOR SCRATCH

Ninja Code Village (OTA, MORIMOTO, KATO, 2016) é um analisador de código de Scratch que foi desenvolvido na *The Open University of Japan*. O objetivo deste analisador é auxiliar os instrutores na avaliação de trabalhos práticos e estimular o ensino de programação para iniciantes. Além de ser um analisador de código, o *Ninja Code Village* tem uma biblioteca com mais de 60 exemplos de funções. Essas funções são, normalmente, utilizadas por crianças em seus programas.

Disponível publicamente online, para analisar um projeto do Scratch no Ninja Code Village, envia-se o arquivo *.sb* ou *.sb2*, como mostrado na Figura 10 à aplicação que executa no navegador de internet. Além de enviar o arquivo do projeto Scratch, pode-se informar algumas diretivas à aplicação, como de qual tipo é o projeto, que pode ser: animação, jogos, arte interativa, música e dança, histórias, detecção de vídeo e outros, e qual é a formação do usuário que criou o código.

Na primeira parte da análise é feito o *parsing* do código e são criadas estatísticas que quais e quantos componentes do Scratch foram utilizados no projeto.

Após geradas as estatísticas é feita uma avaliação do resultado e é gerada uma nota ao pensamento computacional que no Ninja Code Village é dividido em 8 conceitos de programação. Cada conceito recebe sua nota individualmente numa escala de 0 a 4 pontos. Os conceitos são:

- Condicionais;
- Laços;
- Procedimentos;
- Reuso de Procedimentos;
- Dados;
- Eventos;
- Paralelismo;
- Interface de Usuário.

Além das notas, a partir das estatísticas geradas, o Ninja Code Village verifica padrões no código e fornece uma lista de funções presentes em sua biblioteca, que são semelhantes ao código analisado que podem servir como inspiração ao usuário para melhorar ou expandir o seu código.

No *Ninja Code Village* não há a possibilidade de executar em modos de aluno e instrutor e também não é possível gerenciar turmas de alunos.

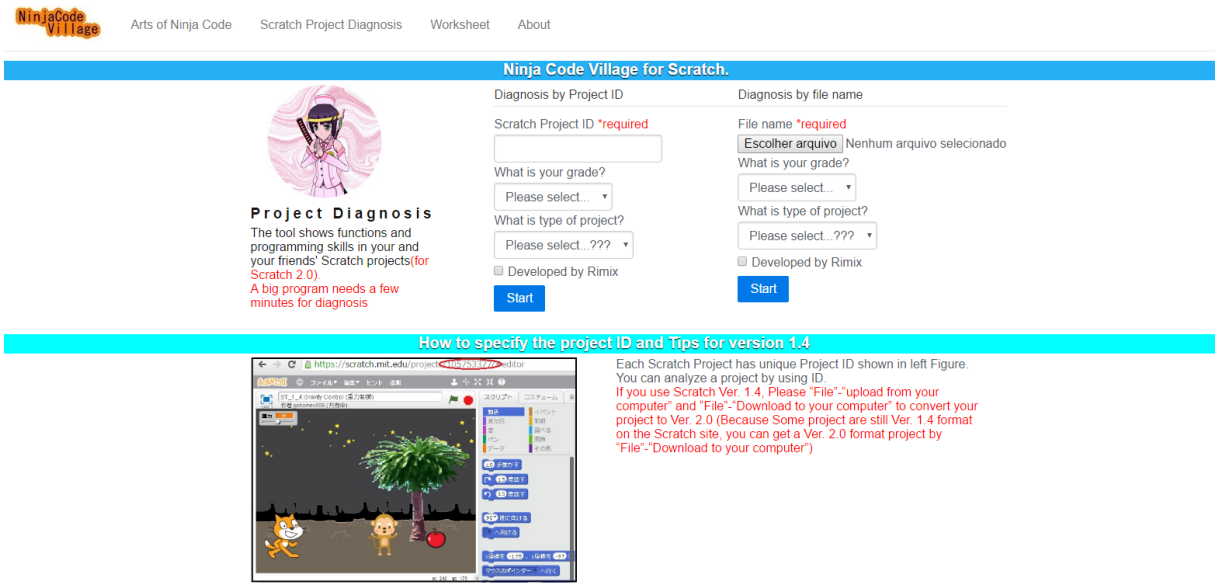


Figura 10 - Tela Inicial do Ninja Code Village

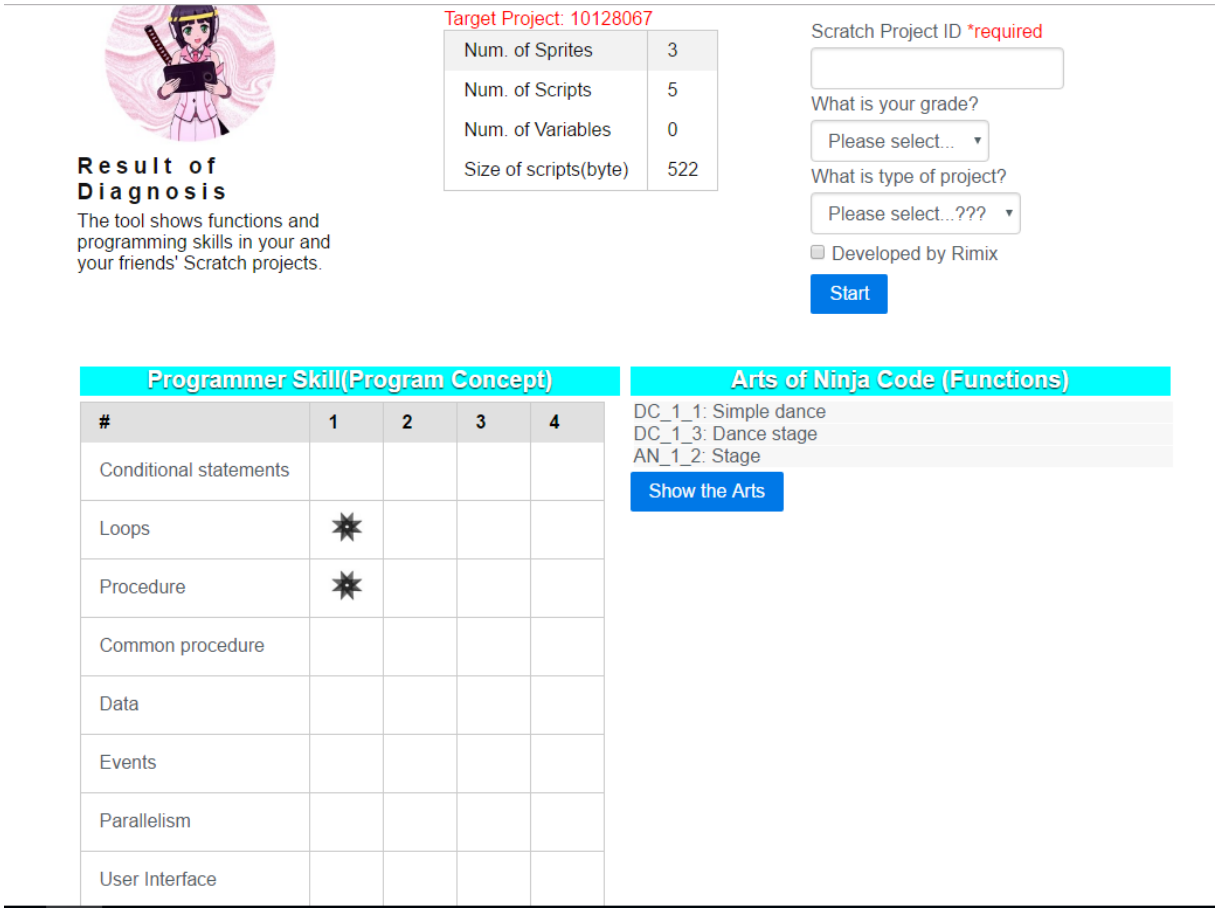


Figura 11 - Tela de Resultado de Análise do Ninja Code Village

3.4 DISCUSSÃO

Observa-se que analisadores estáticos de código para linguagens de programação visuais no contexto de ensino estão quase inexistentes. Na pesquisa englobando qualquer linguagem de programação visual foram encontrados somente dois analisadores para Scratch, sendo que o *HairBall* (BOE, 2013) é uma API que está englobado no Dr. Scratch. Foi encontrado também um resumo sobre um possível analisador para Snap!, porém, sem maiores informações. Não foi encontrado nenhum analisador estático de código para App Inventor, foco do presente trabalho. Portanto é notável a carência de ferramentas de análise de código no contexto de ensino. De forma geral, é perceptível algumas tendências sobre, principalmente, quais conceitos do pensamento computacional são importantes serem analisados por analisadores de código estáticos no contexto de ensino.

Tabela 16 - Comparação Entre Analisadores em Relação as Suas Funcionalidades/Características

	Dr. Scratch	**Ninja Code Village**
Linguagem de Programação Visual	Scratch	Scratch
Licença	Gratuito	Gratuito
Idioma	Português, Inglês, Espanhol, Italiano, Galego, Catalão, Grego e Basco.	Inglês
Notas aos conceitos	Sim	Sim
Comentários	Sim	Não
Exemplos de funções semelhantes	Não	Sim
Gerencia turmas de alunos	Em desenvolvimento	Não
Estatísticas de Projetos Analisados	Sim	Não
Disponibilidade	Online	Online

A Tabela 16 apresenta uma comparação entre características relevantes dos analisadores encontrados conforme apresentado no capítulo 2 da fundamentação teórica.

Tabela 17 - Conceitos do Programação Analisados pelos Analisadores

	Dr. Scratch	Ninja Code Village
Lógica, Condicionais	Sim	Sim
Paralelismo	Sim	Sim
Interatividade com usuário, Eventos	Sim	Sim
Representação de dados	Sim	Sim
Controle de fluxo, loops	Sim	Sim
Sincronização	Sim	Sim
Abstração	Sim	Sim

A Tabela 17 apresenta os conceitos de programação que são incluídos nos diferentes analisadores.

Por meio das análises, observa-se uma tendência de quais conceitos de programação são analisados bem como as principais funcionalidades de um analisador e avaliador de código, como por exemplo a disponibilidade online, gratuidade e fornecer notas e informações de como melhorar seu nível de competência em programação como forma de *feedback* ao usuário.

3.4.1 AMEAÇAS A VALIDADE DO MAPEAMENTO SISTEMÁTICO

Como em toda pesquisa, há potenciais ameaças à validade dos resultados apresentados neste estudo de mapeamento. Existem alguns motivos que podem prejudicar a sua validade, como no caso da inclusão de poucos resultados aceitos tendo em vista que a realização da busca pode ser feita em bases inadequadas, ou seja, em bases que não abordam os assuntos de interesse. Portanto, algumas medidas foram tomadas para mitigar estas ameaças.

Foram utilizadas várias bases digitais para a busca por considerar que estas incluem uma grande quantidade de trabalhos científicos relevantes na área do foco deste estudo. Também foram utilizados os mecanismos de busca Google Scholar e busca no site do App Inventor com o objetivo de encontrar analisadores estáticos de código para linguagens de programação visuais. Tentou-se assim minimizar o risco de não encontrar trabalhos relevantes. Com este objetivo também foram utilizados sinônimos e traduções dos termos de busca maximizando os resultados obtidos, a tradução foi realizada para o Inglês, ampliando o domínio da pesquisa. A abordagem

utilizada para a construção da *string* de busca objetivou a maximização dos resultados obtidos para a pergunta de pesquisa deste trabalho. Porém, termos como "*code analyzer"* ou "*visual programming languages*" produzem muitos resultados irrelevantes para esta pesquisa. Com o objetivo de diminuir os resultados irrelevantes estes termos foram substituídos em algumas buscas por termos específicos, como "*static code analyzer*", "*scratch*" ou "App Inventor" entre outros.

Além disso, a extração e interpretação das informações encontradas foram feitas com cuidado e revisadas por um pesquisador sênior.

4. DESENVOLVIMENTO DO CODEMASTER – APP INVENTOR

Por meio da análise do estado da arte é possível identificar a necessidade de se desenvolver mais ferramentas de análise e avaliação de código para linguagens de programação visual. Portanto, a proposta do presente trabalho é o desenvolvimento de uma ferramenta de análise e avaliação de código voltada especificamente para o App Inventor com foco no ensino de programação para o Ensino Fundamental II. Este trabalho é realizado no contexto do projeto de pesquisa da iniciativa Computação na Escola que visa a ampliação do suporte a avaliação de linguagens de programação baseadas em blocos, desenvolvendo a ferramenta CodeMaster para (semi-)automatizar a avaliação de projetos de programação de App Inventor (foco do presente trabalho) e Snap! (Pelle, 2018). Desta forma visando a integração dos analisadores de código, é realizado em conjunto com Pelle (2018) o desenvolvimento visual e estrutural do servidor destes analisadores.

O objetivo do presente trabalho especifico é desenvolver uma ferramenta de análise de código do App Inventor no contexto de ensino, o CodeMaster - App Inventor. Essa ferramenta deve estar alinhada aos padrões de ensino, análise de código e avaliação abordados na fundamentação teórica (Capitulo 2). A ferramenta deve permitir a análise e avaliação automática dos trabalhos práticos desenvolvidos pelos alunos, fornecendo um *feedback* instrucional.

4.1 MODELO CONCEITUAL

Com base nas ferramentas encontradas no processo de análise do estado da arte e a análise da fundamentação teórica e nas experiências práticas da iniciativa Computação na Escola são identificadas as principais funcionalidades da ferramenta:

- Permitir o upload de projetos do App Inventor versão *nb162a* no formato .aia;
- Analisar o projeto que contém o código da aplicação nos formatos *bky* e *scm*;
- Avaliar um projeto individual retornando um *feedback* instrucional incluindo nota, pontuação total, pontuação por critério e nível ninja;
- Apresentar as avaliações dos projetos ao professor em uma tabela com todos os projetos analisados e podendo exportar esses dados em planilha;
- Apresentar a avaliação do projeto ao aluno.

- Apresentar dados gerais e anônimos sobre as avaliações já realizadas pelo CodeMaster a pesquisadores interessados.

O CodeMaster deve avaliar, com base no Dr. Scratch (DR. SCRATCH, 2017), na rubrica para avaliar *apps* proposta por Sherman (2015) e no *framework* do pensamento computacional (BRENNAN & RESNICK, 2012), os conceitos do código conforme apresentado na Tabela 18.

Tabela 18 - Conceitos do Programação Analisados pelo CodeMaster e Rubrica de Avaliação

Critério	**Nível de Performance**			
	0	**1**	**2**	**3**
Telas	Apenas uma tela com componentes visuais e que seu estado não se altera com a execução do app (tela informativa).	Apenas uma tela com componentes visuais que se alteram com a execução do app.	Pelo menos duas telas e uma delas altera seu estado com a execução do app.	2 ou mais telas e pelo menos 2 delas alteram seus estados com a execução do app.
Interface de Usuário	Utiliza apenas 1 elemento visual sem alinhamento.	Utiliza 2 ou mais elementos visuais sem alinhamento.	Utiliza mais de 5 elementos visuais com 1 tipo de alinhamento.	Utiliza mais de 5 elementos visuais com 2 ou mais elementos de alinhamento.
Nomeação: Variáveis e procedimentos	Nenhum ou poucos nomes são alterado do padrão. (menos do que 10%)	De 10 a 25% dos nomes são alterados do padrão.	De 26 a 75% dos nomes são alterados do padrão.	Mais de 76% dos nomes são alterados do padrão.
Eventos	Nenhum manipulador de evento é usado (ex. On click).	1 tipo de manipuladores de eventos é usado.	2 tipos de manipuladores de eventos são usados.	Mais de 2 tipos de manipuladores de eventos são usados.
Abstração de procedimentos	Não existem procedimentos.	Existe exatamente um procedimento e sua chamada.	Existe mais de um procedimento.	Existem procedimentos tanto para organização quanto para reuso. (Mais chamadas de procedimentos do que procedimentos).
Laços	Não usa laços	Usa “While” (laço simples)	Usa “For each” (variável simples)	Usa “For each” (item de lista)
Condicionais	Não usa condicionais.	Usa apenas “if’s” simples.	Usa apenas “if then else”.	Usa um ou mais “if - else if”.
Operações Lógicas e Matemáticas	Não usa operadores	Usa um tipo de operador.	Usa dois tipos de operadores.	Usa mais de 2 tipos de operadores.

Listas	Não usa listas.	Usa uma lista unidimensional.	Usa mais de uma lista unidimensional.	Usa uma lista de tuplas (map).
Persistência de dados	Dados são armazenados em variáveis ou propriedades de componentes e não tem persistência quando app é fechado.	Usa persistência em arquivo (File ou Fusion Tables).	Usa algum dos bancos de dados locais do App Inventor (TinyDB).	Usa uma base de dados web tinywebdb ou Firebase do App Inventor (firebase ou TinyWebDB).
Sensores	Sem uso de sensores.	Usa um tipo de sensor.	Usa 2 tipos de sensores.	Usa mais de 2 tipos de sensores.
Mídia	Sem uso de mídias.	Usa um tipo de Mídia.	Usa 2 tipos de Mídia.	Usa mais de 2 tipos de mídia.
Social	Sem uso de componentes sociais.	Usa um tipo de componente social.	Usa 2 componentes visuais.	Usa mais de 2 componentes visuais.
Conectividade	Sem uso de componentes de conectividade.	Uso de iniciador de atividades (chama início de outro app no celular).	Uso de conexão bluetooth.	Uso de conexão web, baixo nível.
Desenho e Animação	Sem uso de desenho e animação.	Uso de área sensível ao toque.	Uso de animação com bolinha pré-definida.	Uso de animação com imagem.

Como resultado da análise dos critérios o sistema apresenta uma nota e um nível de competência ao aluno.

Nota. Com base nos conceitos apresentados na Tabela 18, o CodeMaster apresenta uma pontuação para cada critério. A nota final do projeto é definida numa escala de 0 a 10, alinhando a escala de notas tipicamente utilizados no ensino básico no Brasil. Para se obter a nota final, é definida a seguinte fórmula:

$$Nota\ final = \frac{Pontuação\ total\ recebida}{Pontuação\ máxima\ possível} * 10$$

Nível de competência. Para tornar essa nota mais divertida e agradável aos jovens, o resultado é transformado na apresentação de nível de competência representado por faixas de ninjas em cores diferentes. Cada faixa de valores de possíveis notas é mapeado a cores de faixa de ninja, como mostrado na tabela 19.

Tabela 19 - Mapeamento Nota e Faixa Ninja

Nota	Faixa Ninja
0 - 0.9	Branca
1.0 – 1.9	Amarela
2.0 – 2.9	Laranja
3.0 – 3.9	Vermelha
4.0 – 4.9	Roxa
5.0 – 5.9	Azul
6.0 – 6.9	Turquesa
7.0 – 7.9	Verde
8.0 – 8.9	Marrom
9.0 - 10.0	Preta

Feedback. O *feedback* é um conjunto de informações retornadas pela ferramenta após a análise e avaliação de código. Tem o intuito de guiar o aluno no processo de aprendizagem e o professor no processo de ensino. No CodeMaster o *feedback* é apresentado de duas formas, dependendo do modo de uso.

Modo de uso. O CodeMaster tem dois modos de uso. No modo individual o aluno submete seu próprio código à análise sem necessitar de cadastro. No modo individual, o *feedback* inclui:

- A pontuação total e a nota (em termos numéricos e por meio da faixa do ninja usando a imagem do ninja);
- A pontuação por critério de avaliação;
- A rubrica de avaliação.

As pontuações e nota dão ao aluno o *feedback* em termos da performance atingida no desenvolvimento do projeto analisado. A rubrica deve ser utilizada para guiar a melhoria da performance do aluno indicando como poderá melhorar a sua pontuação nos diversos critérios analisados.

No modo professor o mesmo pode submeter um conjunto de projetos App Inventor à ferramenta. Este conjunto de projetos representa uma turma. O feedback é dado em relação a turma toda, indicando por aluno:

- Pontuação total e a nota (em termos numéricos e por meio da cor da faixa do ninja);
- Pontuação por critério de avaliação;
- Médias da turma por critério de avaliação, pontuações totais e notas;
- A rubrica de avaliação.

Por meio dos resultados da avaliação apresentados, o professor analisa o nível de competência que o aluno se encontra, deste modo estará fazendo a avaliação somativa do aluno. Além disso, analisando os resultados globais da turma o professor consegue fazer a avaliação formativa, que diz respeito a avaliação da qualidade da unidade instrucional aplicada ao ensino.

4.2 ANÁLISE DOS REQUISITOS

Com base no modelo conceitual definido são identificados e analisados os requisitos da ferramenta, além de elaborar os fluxos de execução do CodeMaster.

4.2.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais, apresentados na Tabela 20, foram elaborados com base no modelo conceitual apresentando na seção 4.1 que leva em consideração os pontos positivos e deficiências de outros analisadores de código. A Tabela 20 apresenta o mapeamento das atividades do analisador de código por meio dos requisitos funcionais necessários para cobrir o escopo do CodeMaster.

Tabela 20 - Requisitos Funcionais

ID	Requisito	Descrição	Artefato Entrada	Artefato Saída
RF01	Realizar *Upload* de um projeto App Inventor	A ferramenta deve permitir que qualquer usuário envie um projeto App Inventor no formato .aia.	Projeto App Inventor versão *nb156c* selecionado no computador do usuário.	Projeto App Inventor numa matriz de bytes.
RF02	Realizar *Upload* de um conjunto de projetos App Inventor	A ferramenta deve permitir que um professor cadastrado envie um conjunto de projetos (o qual representa uma turma) App Inventor no formato .aia.	Conjunto de projetos App Inventor selecionados no computador do usuário. Cada projeto deve ser nomeado com o nome do aluno para identificação.	Lista de projetos App Inventor cada um no formato .aia. armazenados no banco de dados relacional.
RF03	Descompactar Projeto App Inventor	A ferramenta deve ser capaz de descompactar o projeto App Inventor no formato .aia e localizar o(s) arquivo(s) .bky com o código do projeto.	Projeto App Inventor formato .aia.	Lista de arquivos .bky de cada tela da aplicação.

RF04	Realizar *Parse* do Projeto App Inventor	A ferramenta deve realizar o parse do(s) arquivo(s) .bky encontrado(s) no RF03 gerando uma estrutura de código intermediária.	Lista de arquivos .bky.	Lista de *Document's* (representação interna de código).
RF05	Analisar Projeto App Inventor	A ferramenta deve realizar a análise quantitativa dos elementos definidos para cada conceito conforme detalhada na Tabela 17.	Lista de *Document's.*	Listas de cada tipo de elemento do código definidos na análise de cada conceito
RF06	Avaliar Projeto App Inventor	A ferramenta deve avaliar cada conceito citado na Tabela 17 de acordo com os elementos especificados encontrados no projeto App Inventor e gerar uma nota para cada conceito.	Listas de cada tipo de elemento do código.	Uma resposta de avaliação formada pela nota para cada conceito, um nível de competência e um feedback.
RF07	Apresentar Avaliação do projeto App Inventor de forma detalhada	A ferramenta deve apresentar na interface de usuário a avaliação detalhada do seu projeto. 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.
RF08	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.	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 a análise.
RF09	Realizar Cadastro de Professor	A ferramenta deve permitir o cadastro de professor por meio de sua interface Web.	Nome completo, e-mail, senha, instituição, cidade, estado, país.	Insere cadastro no banco de dados. Interface de usuário exibe mensagem de sucesso, ou mensagem de erro caso algo dê errado.
RF10	Realizar *Login* de Professor	A ferramenta deve permitir que o professor autentique sua identidade por meio de *login.*	E-mail e senha.	Redirecionado para página principal do professor ou interface com usuário exibe mensagem de erro caso não consiga autenticar.
RF11	Realizar *Logout* de Professor	A ferramenta deve permitir que o professor encerre a sessão atual de autenticação de sua identidade por meio de *logout*.	*Login* do professor na sessão atual.	Encerramento da sessão atual e redirecionamento para página inicial da ferramenta.
RF12	Manter Registro de Avaliações do Professor	A ferramenta deve manter em seu banco de dados todos os registros de avaliações feitas por professores por turma	*Login* do professor e projetos submetidos na sessão atual.	Sistema armazena em seu banco de dados todas as avaliações de projetos que o professor submeteu na sessão atual.
RF13	Manter Registro de Avaliação de Usuário Anônimo	A ferramenta deve manter em seu banco de dados o registro de uma avaliação feita por usuário anônimo.	Projeto submetido por usuário anônimo.	Sistema armazena em seu banco de dados toda avaliação de projeto que qualquer usuário anônimo tenha submetido.
RF14	Apresentar todas Avaliações do Professor	A ferramenta deve apresentar em sua interface com o	*Login* do professor e todas avaliações presentes no banco de	Interface com usuário apresenta em tabela todas as avaliações (notas de cada

		usuário todas as avaliações já feitas pelo professor.	dados de projetos submetidos pelo professor autenticado.	conceito, nota final, nível) presentes no banco de dados do professor autenticado.
RF15	Apresentar Estatísticas de Avaliações	A ferramenta deve apresentar em sua interface com o usuário estatísticas sobre todos os projetos App Inventor já analisados.	Todas avaliações presentes no banco de dados de projetos já submetidos a análise na ferramenta.	Interface com usuário apresenta informação quantitativa de cada elemento analisado pelo Inventor Analyzer, apresenta média de notas finais, média de notas de cada conceito.

4.2.2 REQUISITOS NÃO-FUNCIONAIS

A Tabela 21 apresenta os requisitos não-funcionais da ferramenta.

Tabela 21 - Requisitos Não Funcionais

ID	Requisito	Descrição
RNF01	Sistema Web	A ferramenta deve ser acessada via navegador Web com conexão à internet. Navegadores compatíveis: Google Chrome versão 56.0.2924.87; Mozilla Firefox versão: 51.0.1; e Microsoft Edge versão 38.14393.0.0.
RNF02	Linguagem de Programação Java	A ferramenta deve ser desenvolvida na linguagem de programação Java, pois é uma linguagem muito conhecida facilitando a manutenção do sistema por profissionais qualificados.
RNF03	Políticas de Privacidade	O rodapé da interface do usuário da ferramenta deve conter as informações de políticas de privacidade adotadas ou *link* para página que contenha tais informações.
RNF04	Termos de Uso	O rodapé da interface do usuário da ferramenta deve conter as informações de termos de uso adotados ou *link* para página que contenha tais informações.
RNF05	Usabilidade	•Eficácia: 90% dos usuários da avaliação por painel de especialistas devem conseguir completar a tarefa de analisar um ou múltiplos códigos sem assistência. •Satisfação: Pontuação total de SUS: 80 pontos.
RNF06	Identidade Visual da CnE	A interface de usuário deve ter padrões de identidade visual definidos pela Iniciativa Computação na Escola.
RNF07	Informação de Progresso da Análise	A interface de usuário deve mostrar informação de progresso da análise de projeto em execução.
RNF08	Desempenho	A ferramenta deve analisar um conjunto de até 30 projetos em menos de 30 segundos num computador com no mínimo: processador com 2 núcleos e 2gb de ram; além de uma conexão de internet a cabo de pelo menos 5mbps.
RNF09	Projeto App Inventor	O sistema deve analisar projetos App Inventor da versão *nb162a*.
RNF10	Internacionalização	O sistema deve conter a opção de textos em português e inglês.
RNF11	Extensibilidade	O sistema deve permitir que novos critérios de análise sejam acrescentados ao sistema no futuro.

4.3 CASOS DE USO

Analisando o modelo conceitual da seção 4.1 e o requisitos foram elaborados os casos de uso do sistema, bem como seus principais fluxos de execução.

Existem elementos computacionais e usuários que interagem com o sistema e são parte fundamental da aplicação como um todo, esses elementos são os atores do sistema.

- [A01] Aluno – Envia atividade para avaliação, vê resultado da avaliação.
- [A02] Professor – Envia conjunto de projetos para avaliação, vê resultado das avaliações por turma.
- [A03] Pesquisador – Acessa as estatísticas de todas as avaliações presentes no banco de dados.

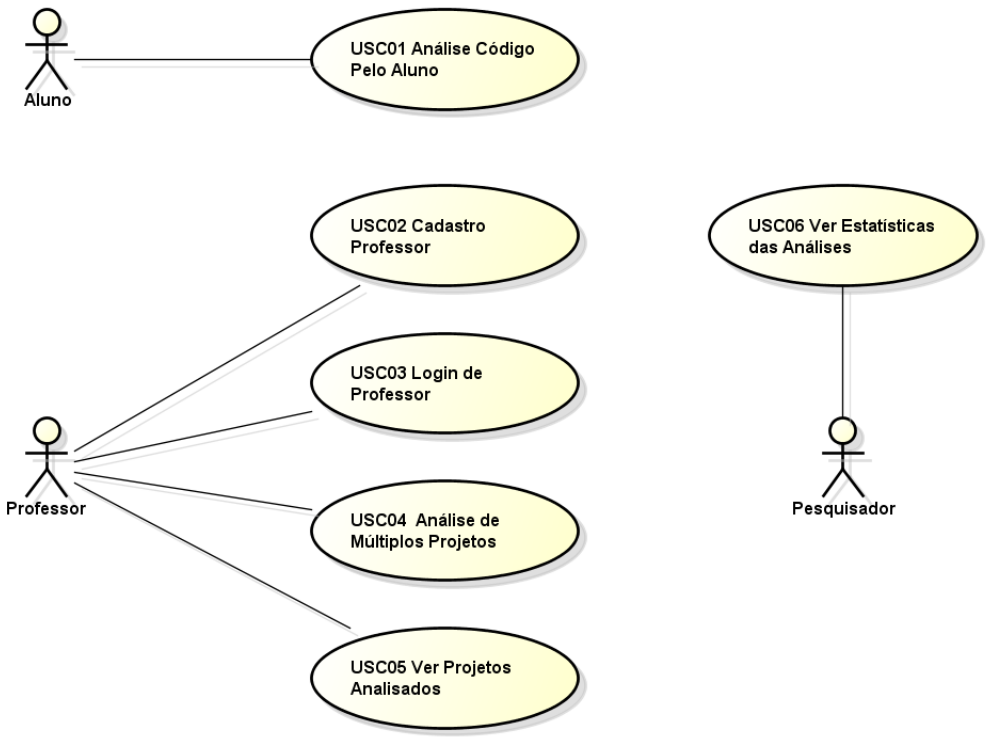


Figura 12 - Diagrama de Casos de Uso do CodeMaster

Caso de Uso 1 – USC01 Análise Código Pelo Aluno

Atores:

- Aluno

Fluxo principal:

1. Aluno acessa o site do CodeMaster.
2. Aluno clica no item de menu “Aluno”.
3. Aluno clica em escolher arquivo e escolhe projeto em seu computador.
4. Aluno clica em “Avaliar” e sistema retorna o resultado da análise do projeto.

Fluxo de Exceção:

FE01 – Sistema não consegue analisar o projeto enviado e uma página informando que um “Erro Ocorreu” com a opção de voltar para tela inicial é exibida.

FE02 - Aluno clica em avaliar sem escolher projeto. Sistema informa uma mensagem: “Nenhum projeto foi selecionado para análise! ”.

Caso de Uso 2 – USC02 Cadastro Professor

Atores:

- Professor

Fluxo Principal:

1. Professor acessa site do CodeMaster
2. Professor clica no menu superior no botão “professores”
3. Professor clica na opção “Ainda não dou cadastrado” abaixo dos campos de login e senha.
4. Professor informa, nome, e-mail, senha, instituição de ensino, disciplina, cidade, estado e clica em cadastrar.
5. Professor recebe e-mail para validar conta.
6. Professor abre link recebido no e-mail e uma mensagem de conta confirmada é exibida
7. Página da área de login de professor é exibida.

Fluxo Alternativo:

4a. Professor não preenche um dos campos de texto e o sistema apresenta a mensagem “Todos os campos devem ser preenchidos”.

5a. E-mail informado já existe cadastrado, sistema apresenta a mensagem “Já existe uma conta com este e-mail, verifique o e-mail informado! ”.

Caso de Uso 3 – USC03 Login de Professor

Atores:

- Professor

Pré-condição:

- USC02

Fluxo Principal:

1. Professor acessa o site do CodeMaster.
2. Professor clica no botão professores no menu superior da tela.
3. Professor informa seu e-mail e senha nos campos de login e senha e clica em Login.
4. O sistema autentica o professor no sistema.
5. O sistema autentica o professor e é redirecionado à página principal do modo professor (página de avaliação).

Fluxo de Exceção:

FE02 - O e-mail e senha do professor não são identificados. O sistema informa uma mensagem: “Usuário e senha não identificados pelo sistema”.

Caso de Uso 4 – USC04 Análise de Múltiplos Projetos

Atores:

- Professor

Pré-condição:

- USC03

Fluxo Principal:

1. Professor digita o nome da turma e seleciona o tipo de projeto que deseja avaliar e clica em próximo.
2. Professor seleciona os conceitos que deseja avaliar.
3. Professor escolhe múltiplos arquivos de seu computador que correspondem a uma turma.
4. Professor clica em avaliar e recebe o resultado da análise de todos os projetos em uma tabela.

Fluxo de Exceção:

FE01 - Professor clica em avaliar sem escolher projetos. Sistema informa uma mensagem: “Nenhum projeto foi selecionado para análise!”.

Caso de Uso 5 – USC05 Ver Projetos Analisados

Atores:

- Professor

Pré-condição:

- USC02

Fluxo Principal:

1. Professor clica em “Turmas” no menu superior da área de professores.
2. Professor seleciona a turma que deseja ver as análises.
3. Sistema consulta no Banco de Dados e retorna o resultado das análises da turma selecionada.

Fluxo de Exceção:

FE01. Nenhuma turma foi previamente cadastrada pelo professor. Sistema apresenta mensagem: “Nenhuma turma foi encontrada”.

Caso de Uso 6 – USC06 Ver Estatísticas das Análises

Atores:

- Pesquisador

Fluxo Principal:

1. Pesquisador acessa o site do CodeMaster.
2. Pesquisador clica no menu superior em “Admin”, sistema faz consulta no Banco de Dados e retorna dados estatísticos (média geral de notas, média geral de pontuações por critérios de análise, quantidade de projetos avaliados, lista e frequência dos blocos de códigos mais utilizados nos projetos).

Fluxo de Exceção:

FE01. Nenhum projeto foi analisado anteriormente. Sistema apresenta mensagem: “Nenhuma análise encontrada em nossa base de dados”.

4.4 MODELAGEM E IMPLEMENTAÇÃO DO CODEMASTER

O sistema de análise de código proposto foi desenvolvido usando a linguagem de programação Java com JSP (linguagem Java para desenvolvimento Web). Essas tecnologias foram escolhidas pelo fato do aluno e equipe de desenvolvimento ter conhecimento prévio delas e por serem as tecnologias disponíveis na infraestrutura de servidor utilizado.

Foi definido o modelo de arquitetura do sistema objetivando a separação das camadas de apresentação e analise em diferentes módulos, tornando a aplicação escalável a longo prazo, permitindo a conexão direta de outras aplicações ao serviço de análise de código no futuro. A Figura 13 apresenta um diagrama de componentes do CodeMaster. O sistema é dividido em 3 grandes blocos, um container web com módulo de apresentação e acesso a banco de dados, um container web exclusivo para o módulo de análise e avaliação de código, e o próprio navegador de internet do usuário onde a interface gráfica do sistema é exibida.

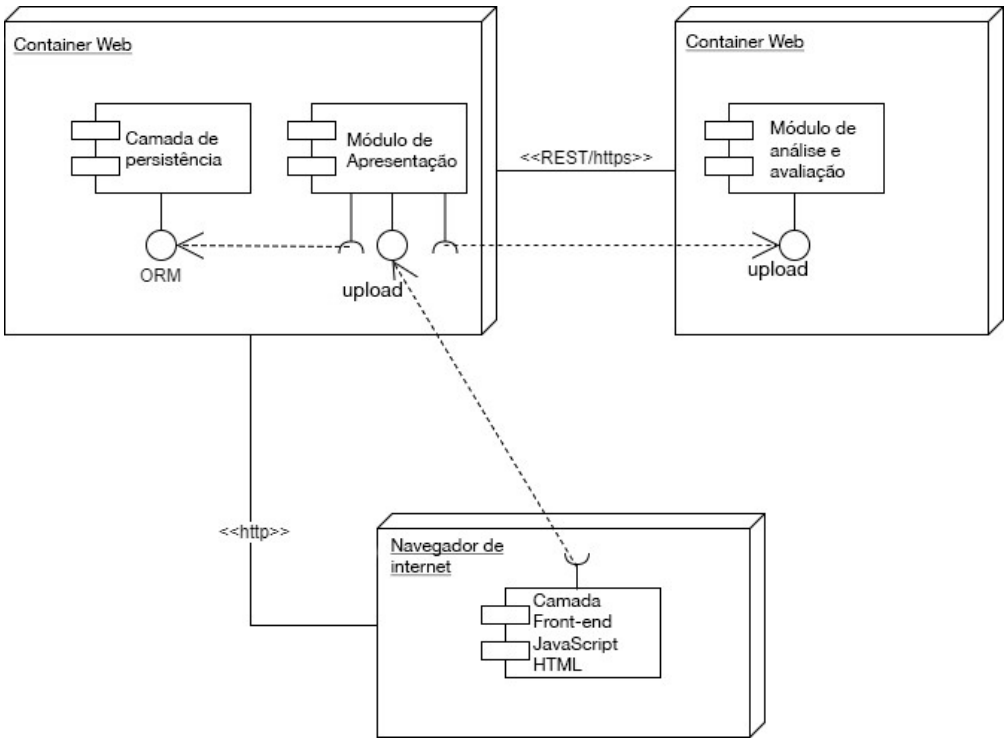


Figura 13 - Diagrama de Componentes do CodeMaster

O módulo de avaliação é um serviço web *REST* responsável no sistema por receber o projeto, receber configurações e retornar o resultado da análise de acordo

com o modelo de avaliação definido na seção 4.1. *REST* (RODRIGUEZ, 2008) é uma arquitetura que permite a separação de sistemas web em módulos (serviços). Para implementar esse serviço o CodeMaster utiliza o *framework* Jersey (JERSEY, 2017) que usa a API JAX-RS (JAX-RS API, 2017) abstraindo os detalhes de baixo nível da implementação de comunicação entre os servidores e simplifica a implementação de um serviço *REST*. A escolha desse *framework* foi feita pela simplicidade de uso, por ser adequado ao que é proposto no CodeMaster e pela grande quantidade de material disponível, auxiliando na implementação do sistema.

O módulo de apresentação é o responsável pelo controle da interface de usuário, o registro de professores e turmas, a submissão de projetos e a apresentação de resultados tanto para professores quando alunos. O componente de *front-end* utiliza as tecnologias JSP, Java Script, HTML5 e CSS3 comuns no desenvolvimento de páginas web.

4.4.1 ARQUITETURA DO MÓDULO DE AVALIAÇÂO

O módulo de avaliação foi projetado para que seja extensível no sentido de poder receber mais analisadores de código de outros sistemas (p. ex. Scratch) no futuro e também no sentido de expandir as análises dos analisadores presentes, permitindo que novos conceitos do pensamento computacional sejam adicionados ou, os já existentes, alterados de forma simples e baixo esforço.

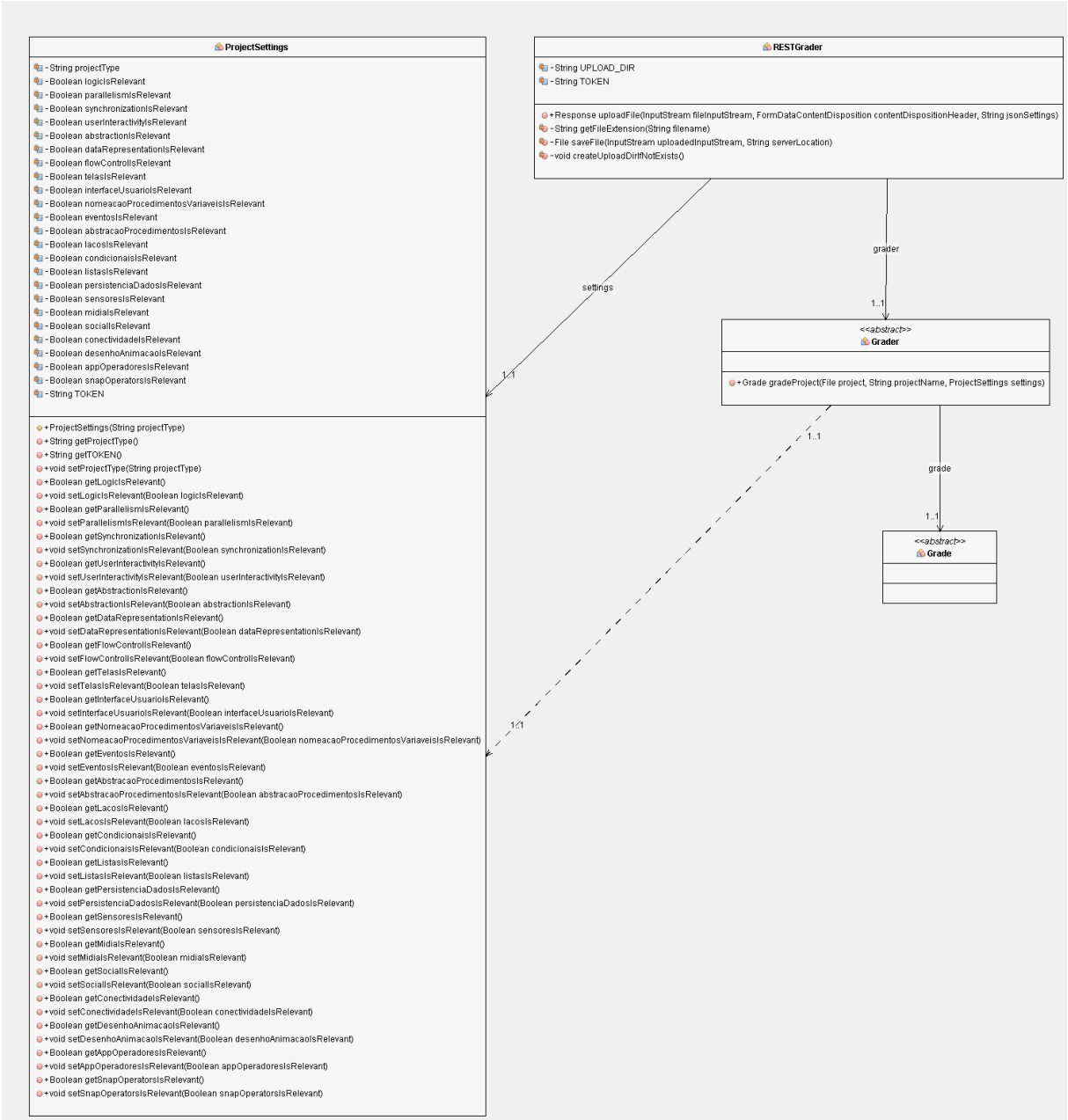


Figura 14 - Diagrama de Classes do Serviço REST

A Figura 14 apresenta as classes principais e de controle do serviço REST do CodeMaster. O serviço recebe o projeto e o arquivo de configuração e deve coordenar a execução da análise. Foi definida uma classe abstrata *Grader*, e qualquer instância de analisador que estiver disponível no sistema deve expandir essa classe e implementar o método *gradeProject* que recebe o projeto e a configuração. Da mesma forma, deve retornar ao coordenador do serviço uma instância da classe *Grade* com a avaliação completa do projeto recebido.

A classe *ProjectSettings* é a classe de configuração do analisador, ela baliza a análise de código determinando quais conceitos do pensamento computacional devem ser analisados, evitando o processamento desnecessário de informação por parte do analisador.

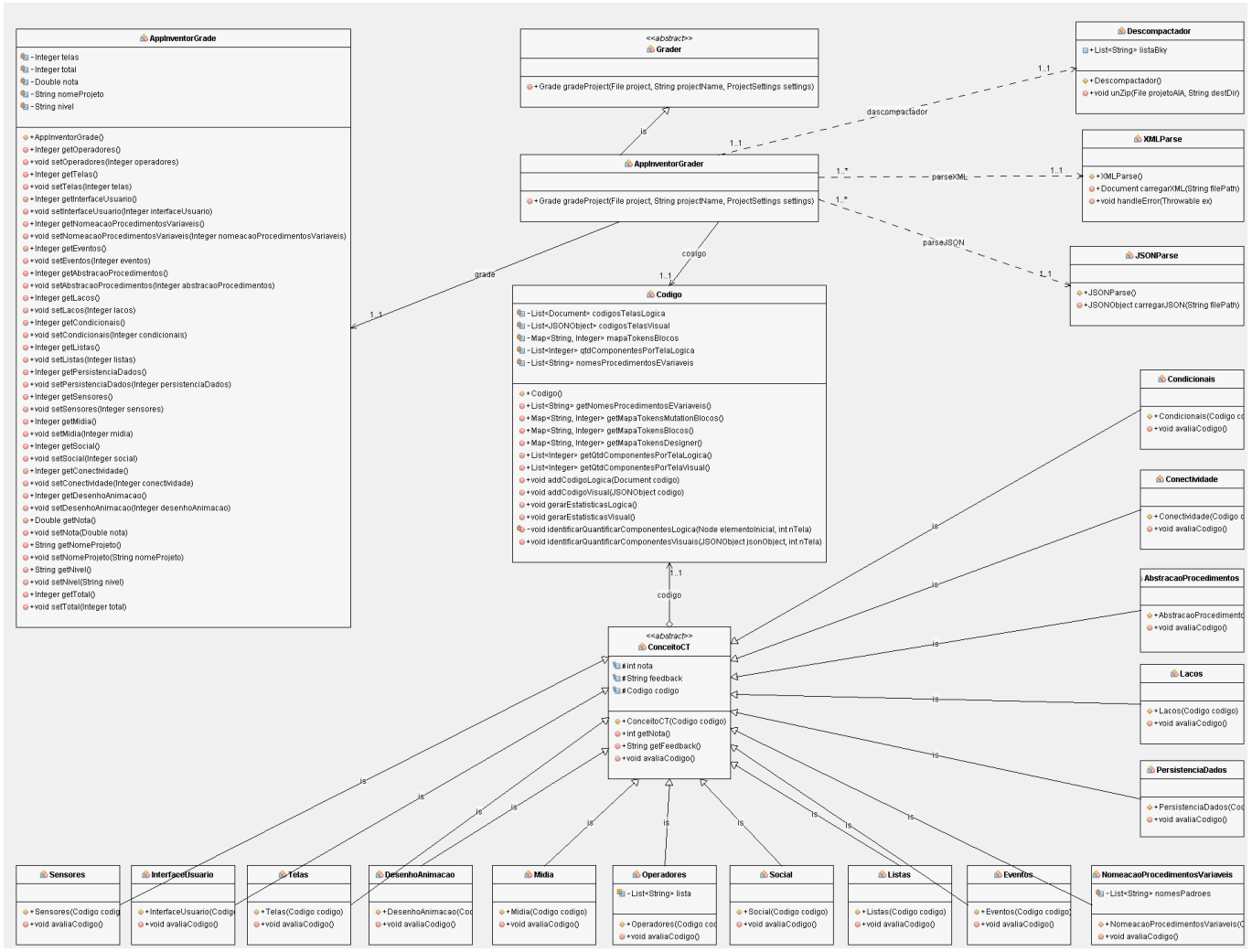


Figura 15 - Diagrama de Classes do analisador e avaliador do App Inventor

A Figura 15 apresenta o diagrama de classes do analisador de código do App Inventor implementado no sistema CodeMaster. A classe *AppInventorGrader* implementa a classe abstrata *Grader* e é a classe principal da implementação. Ela é inicialmente responsável por receber o projeto no formato .aia e enviar para o *Descompactador* que retornará os arquivos *bky* e *scm* de cada tela do projeto App Inventor. Para cada tela criada no app é gerado um arquivo no formato *bky* e um arquivo no formato *scm.* O arquivo *bky* encapsula um estrutura *xml* com todos os blocos de programação utilizados na lógica do app, já o arquivo *scm* encapsula uma estrutura *json* que contém todos os componentes visuais utilizados no app. Estes

arquivos contém os códigos a serem analisados, portanto, deve ser feito o parse, para isso é utilizada a biblioteca *JSON Simple* de código aberto para o parse dos arquivos *json* e bibliotecas nativas do Java para o parse dos arquivos *xml*. Esta etapa é feita pelas classes *XMLParse* e *JSONParse* que constroem estruturas de dados em memória com o código e retornam. Por sua vez essas estruturas de dados são passadas para a classe *Codigo* que, por meio de um algoritmo, percorrerá essas estruturas e contabilizara todos os blocos utilizados, gerando estatísticas sobre os componentes do projeto recebido.

ConceitoCT é uma classe abstrata que recebe uma instância da classe *Codigo* onde contém todas as estatísticas do projeto. Cada conceito definido na seção 4.1 estende essa classe *ConceitoCT* e implementa o método *avaliaCodigo* retornando com base nas estatísticas geradas uma pontuação de 0 a 3 para tal conceito. Por fim, todas as pontuações são calculadas de modo a gerar uma pontuação, nota final e nível ninja e retornadas encapsuladas na classe *AppInventorGrade*.

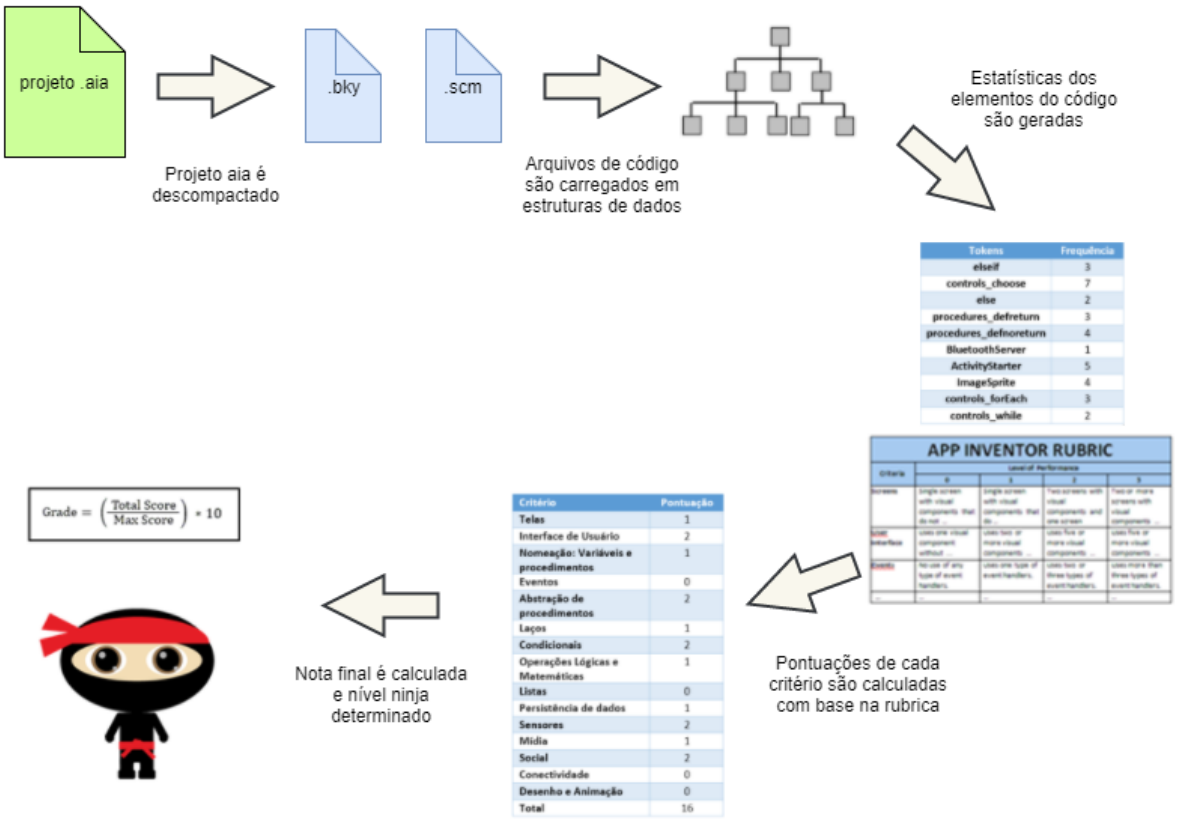


Figura 16 - Sequência da Análise de Código CodeMaster

A Figura 16 apresenta a sequência do processo de análise e avaliação de projetos App Inventor. Quando o módulo de avaliação recebe o arquivo de projeto e o arquivo de configurações, os mesmos são enviados ao avaliador do App Inventor. O primeiro passo do avaliador é descompactar o projeto e obter apenas os arquivos com extensão *bky* e *scm*, que contém as informações de código relevantes para a análise.

Estes arquivos com os códigos do projeto App Inventor são carregados em memória em estruturas de dados próprias das bibliotecas utilizadas. Feito isso, começa o processo de geração de estatísticas de cada componente presente no código, nesse momento é contabilizado quanto de cada componente foi utilizado. Após a finalização da geração de estatísticas, as quais ficam armazenadas em mapas que relacionam nome do componente a sua quantidade, para cada critério de avaliação marcado como verdadeiro na classe de configuração *ProjectSettings* recebido, é feito o cálculo da pontuação obtida de 0 a 3 com base na rubrica da seção 4.1. Com todas as pontuações dos critérios relevantes prontas, o cálculo final da nota é feito e por consequência o nível da faixa ninja é também gerado. O resultado final é armazenado numa instância da classe *AppInventorGrade* que é retornada ao módulo de apresentação que armazena num banco de dados e apresenta o resultado para o usuário.

4.4.2 ARQUITETURA DO MÓDULO DE APRESENTAÇÃO

O módulo de apresentação é responsável por receber o projeto do usuário por meio da interface web, enviar o projeto ao analisador e apresentar o resultado da avaliação ao usuário. Além disso, este módulo é responsável pelo cadastro de professores, a autenticação do professor perante o sistema, envio de e-mail para confirmação de cadastro, gerenciar o idioma e gerenciar todo o armazenamento de projetos, turmas, professores e administradores no banco de dados.

Para possibilitar a implementação deste módulo em Java foi utilizada a tecnologia JSP para a criação das páginas web que o usuário vê. Cada página criada com JSP é transformada em um *servlet* Java que receberá todas as requisições vindas do usuário. Por meio das requisições aos *servlets* os fluxos de execuções acontecem. Cada caso de uso gera fluxos de execuções diferentes, gerados por requisições aos *servlets*.

Todos os fluxos que dependem da análise dos projetos invocam a classe *HTTPClient* que faz o envio do arquivo de projeto e recebe a resposta do serviço de análise. Os fluxos que dependem do banco de dados, tanto para salvar informações ou para consultas, chamam métodos da classe *Control*, que acessam o banco de dados através dos respectivos objetos de acesso aos dados (DAOs). Os fluxos que dependem de envio de e-mail chamam o método da classe *Mail* que faz o envio de e-mail simples.

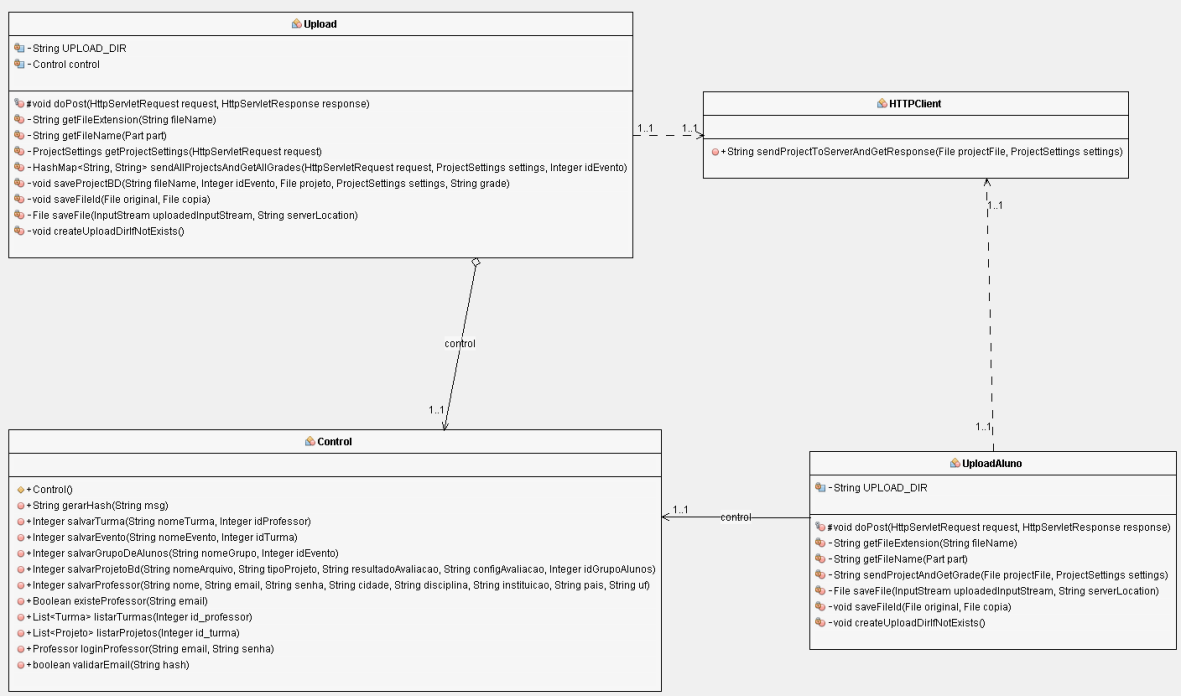


Figura 17 - Diagrama de Classes Reduzido da Análise de Código do Módulo de Apresentação

No diagrama de classes apresentado na Figura 18 é possível observar que as classes *Upload* e *UploadAluno* tratam as requisições vindas dos *servlets* de avaliação de projetos da área de professor e de avaliação da área de aluno respectivamente. Neste diagrama é apresentado apenas as classes principais que interagem com as classes de upload, como a *HTTPClient* responsável pela comunicação com o módulo de avaliação e a *Control* responsável por salvar os projetos avaliados no banco de dados.

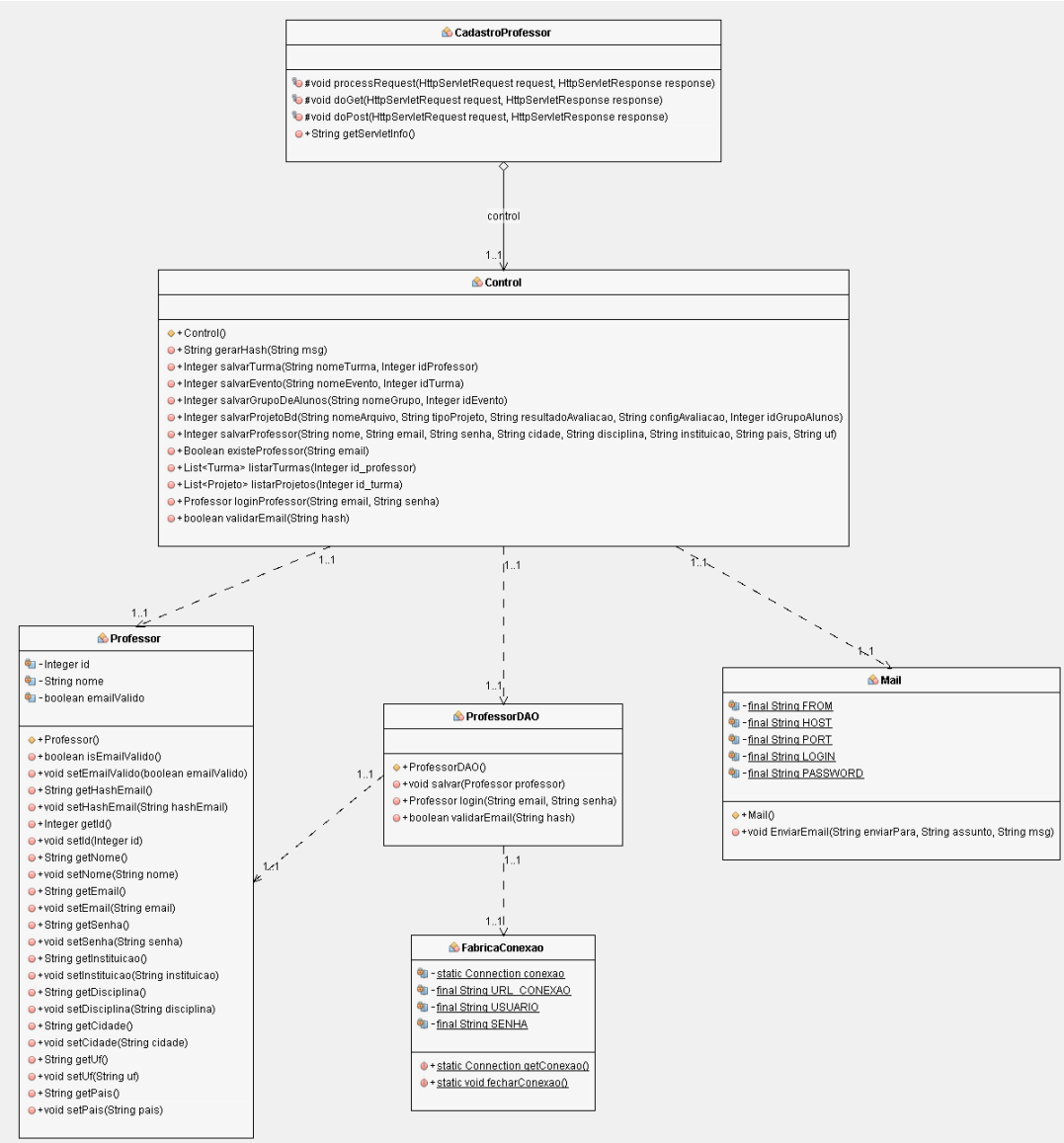


Figura 18 - Diagrama de Classes do Cadastro de Professor

O diagrama apresentado na Figura 18 ilustra as classes responsáveis pela armazenagem de informação no banco de dados por meio do exemplo de cadastro de professor. A classe *CadastroProfessor* recebe a requisição de um novo cadastro, para isso ela chama métodos responsáveis da classe *Control* que armazenam os dados por meio do objeto de acesso aos dados do professor (*ProfessorDAO*), após o cadastro concluído um e-mail com o link de confirmação é enviado ao endereço de e-mail do professor cadastrado.

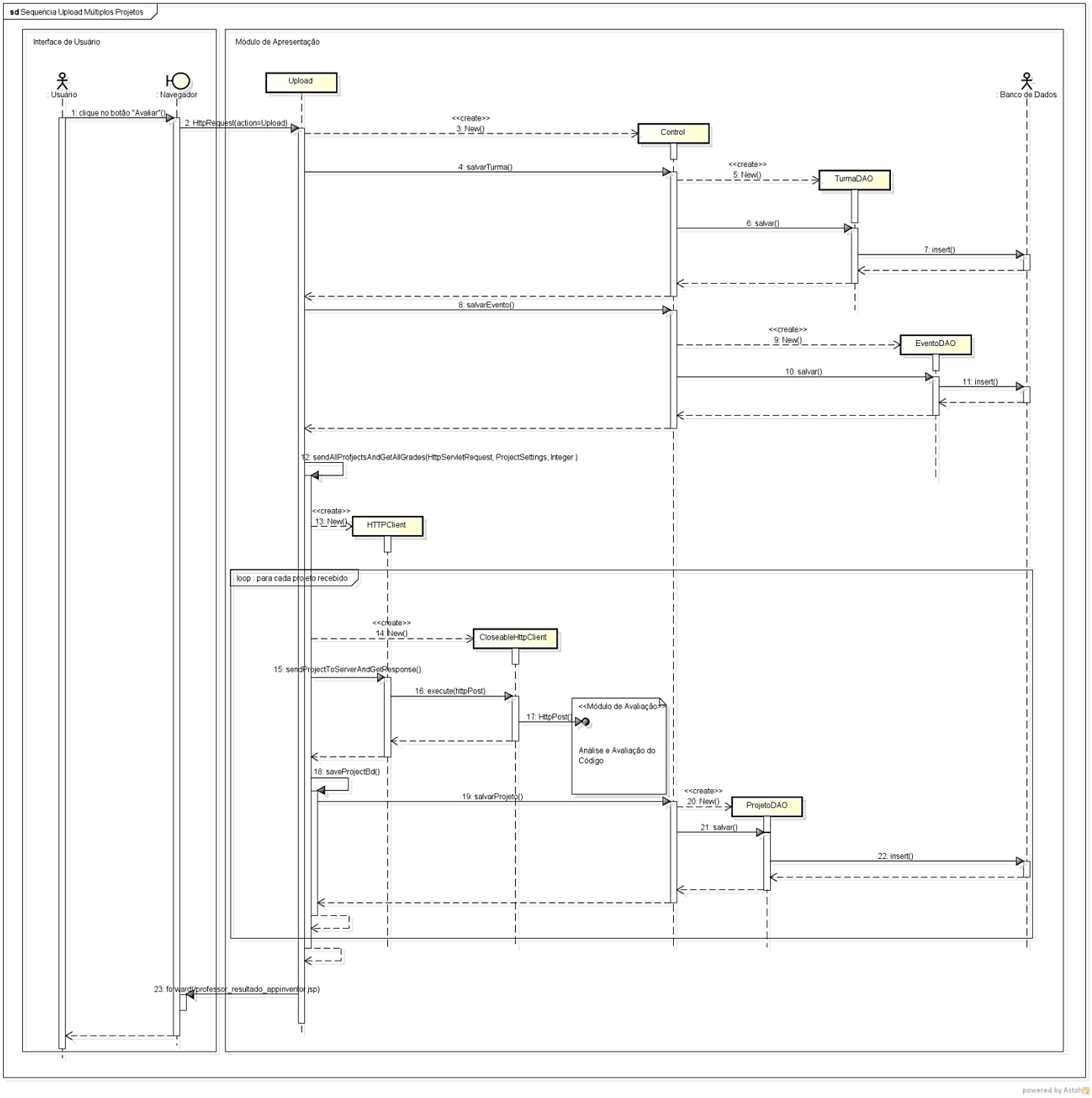


Figura 19 - Diagrama de sequência de upload de múltiplos projetos

A Figura 19 apresenta a sequência de mensagens trocadas pelo módulo de apresentação para a avaliação de múltiplos projetos enviados por um professor. O fluxo de execução para análise de um único projeto enviado por um aluno é análogo, alterando apenas a presença de um laço.

A camada de persistência é utilizada apenas pelo módulo de apresentação. Esta camada utiliza um banco de dados MySQL (MYSQL, 2017). O MySQL permite a criação de bancos de dados relacionais. O banco de dados do CodeMaster foi projetado de forma que sua estrutura ficasse simples e objetiva, porém, que permita futuras implementações no sistema sem que os dados já armazenados se tornem inconsistentes.

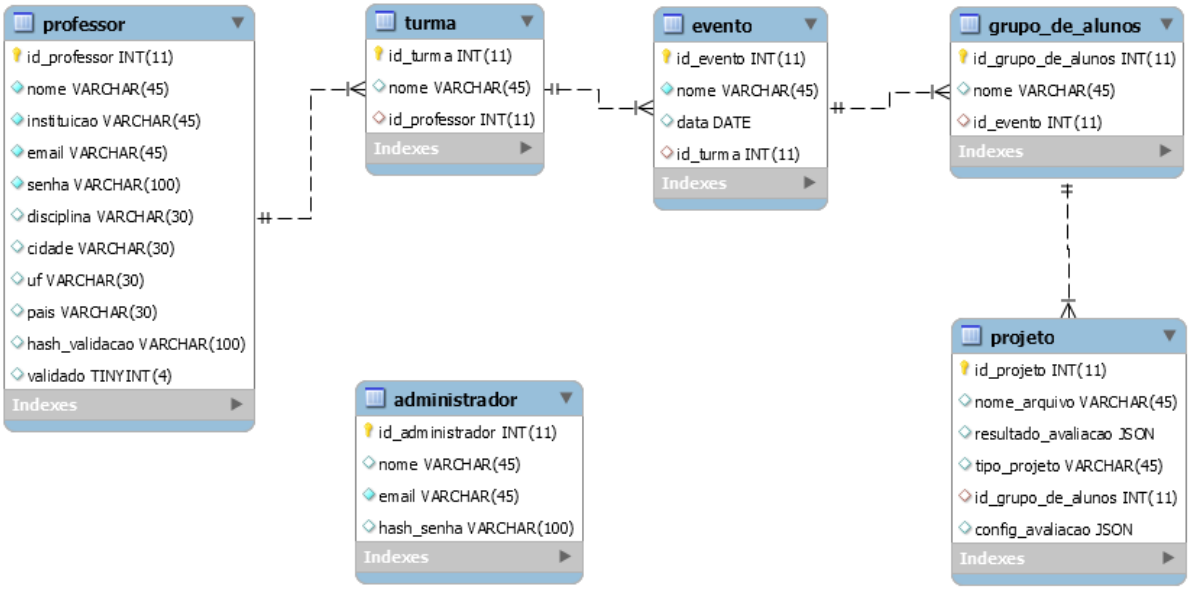


Figura 20 - Diagrama Entidade Relacionamento do Banco de Dados CodeMaster

A Figura 20 apresenta as tabelas presentes no banco de dados do CodeMaster bem como as suas relações.

Os dados de cadastro de professores são armazenados na tabela *professor*, bem como login e senha para acesso ao modo professor do CodeMaster. Para uma maior segurança, a senha do professor cadastrado não é salva em formato original, mas em sim um *hash* da senha, obtido por meio do algoritmo SHA-256, desta forma é possível a partir da senha obter o *hash*, mas a partir do *hash* é impossível obter a senha. Isso dificulta a descoberta da senha dos usuários por pessoas mal-intencionadas que possam vir a acessar o banco de dados. Além dos dados comuns cadastrados como nome, cidade, estado, disciplina, dois campos são utilizados para a validação do e-mail do professor. O *hash_validacao* armazena o *hash* do e-mail cadastrado, esse *hash* é enviado por e-mail para a validação da conta do professor e

o campo *validado* que armazena uma variável do tipo booleana e informa se o professor validou o seu cadastro por meio do link de confirmação recebido por e-mail.

A tabela *turma* armazena um ID da turma criada, o nome que o professor deu para a turma e uma chave estrangeira que relaciona a tabela professor e informa qual é o professor de determinada turma.

É possível que futuramente se deseje acrescentar a informação de eventos que determinada turma participa ao CodeMaster, cada evento será armazenado na tabela *evento* que contém um nome e uma data do respectivo evento.

A tabela *grupo_de_alunos* armazena a informação do nome do grupo de alunos ou o nome de um aluno em especifico que criou determinado projeto. O nome do grupo de alunos ou aluno é determinado por meio do nome do arquivo de projeto enviado ao sistema. Portanto, no momento do envio do projeto para avaliação ao CodeMaster é importante que o nome do projeto contenha esta informação (o. ex.: MariaSilva.aia). Essa informação facilita a diferenciação dos alunos na exibição do resultado da análise e avaliação de projetos de uma turma.

A tabela *projeto* armazena as informações de projetos avaliados. Ela contém um ID de projeto que é utilizado como novo nome de arquivo para armazena-lo no servidor, o nome original do projeto avaliado, o resultado da avaliação em JSON salvo a partir da classe *Grade*, a configuração da análise em JSON salva a partir da classe *ProjectSettings,* o tipo de projeto (App Inventor ou Snap!) e uma chave estrangeira relacionando o projeto a um grupo de alunos. Se o projeto for submetido à avaliação do CodeMaster por um aluno, de forma anônima, o campo que relaciona o projeto a um grupo de alunos ou aluno é nulo.

A tabela *administrador* ainda não é utilizada pelo CodeMaster, ela existe para possibilitar o cadastro de possíveis administradores de sistema e pesquisadores que poderão ter acesso a páginas de estatísticas e informações especificas do sistema que o usuário comum não deverá possuir.

4.4.3 TESTES DE UNIDADE DO SOFTWARE

Os testes de unidade têm a finalidade de testar partes específicas do código, geralmente são testes em nível de componentes ou classes do software. Os testes de

unidade foram performados no CodeMaster – App Inventor em paralelo com o seu desenvolvimento.

Os primeiros testes realizados foram nas funções que analisam e avaliam os 15 critérios definidos na rubrica. Para cada critério há uma função correspondente no código. Cada função foi testada inicialmente por projetos App Inventor criados de forma que contivessem apenas blocos específicos do critério analisado. Após resultados satisfatórios na avaliação de cada critério separado, foram feitos testes com aplicativos funcionais desenvolvidos pela iniciativa Computação na Escola, como por exemplo o jogo do mosquito, já integrando todos os critérios e gerando uma nota final.

Após os testes no avaliador de código propriamente dito, testes na integração dos avaliadores App Inventor e Snap! e servidor do módulo de avaliação foram feitos. Para isso foi criada uma tela de upload de arquivo temporária neste módulo. Com isso foi possível testar o envio de arquivos de projetos das diferentes linguagens, testar o algoritmo de seleção do avaliador que deve ser alocado pelo sistema e testar o recebimento do resultado das avaliações.

Acompanhando o desenvolvimento incremental do módulo de apresentação, os testes foram sendo feitos. Primeiramente foi desenvolvida a função de envio de projetos únicos, portanto, a função de envio para alunos. Deste modo foi possível testar: o *upload* de arquivos de projetos no módulo de apresentação, o envio do projeto ao módulo de avaliação, o recebimento do resultado da avaliação e a apresentação do resultado ao usuário. Após os testes com apenas um projeto terem retornado resultados satisfatórios, testes nas funções de *upload* de múltiplos projetos foram performados.

Por fim os testes a camada de persistência foram feitos. Foram feitos testes nas funções de cadastro de professores, de login de professores, ao salvamento de turmas de alunos e projetos. Também foram feitos testes na função de envio de e-mail e validação de conta de professores.

4.4.4 EXPANDINDO O CODEMASTER

O CodeMaster foi projetado para que as expansões sejam fáceis de serem implementadas. Nesta seção é apresentado como adicionar um novo critério de

análise ao analisador e avaliador de códigos do App Inventor e como adicionar um novo analisador e avaliador ao CodeMaster.

Adicionando novo critério de avaliação.

Adicionar um novo critério de avaliação diz respeito a expandir as avaliações do analisador e avaliador de projetos App Inventor atualmente disponível. Futuramente, talvez se queira adicionar mais um critério de avaliação à rubrica, como por exemplo, avaliar a usabilidade do aplicativo. Para isso não é necessário criar um novo analisador, fazendo algumas alterações consegue-se expandir o sistema para atender o novo critério.

No módulo de apresentação deve ser adicionado mais um *checkbox* a tela de professor que contém os critérios que se deseja analisar. Nas telas de resultado de aluno e professor deve ser adicionado uma nova coluna para o novo critério a ser adicionado. Outra alteração neste módulo é na classe *ProjectSettings* que deve receber uma nova variável booleana com o novo critério. A classe *Upload* é capaz de obter o valor da variável vindo da requisição do usuário. A classe *AppInventorGrade* deve receber uma nova variável que receberá a pontuação do critério.

No módulo de análise e avaliação, o *ProjectSettings e AppInventorGrade* devem receber as mesmas alterações que as respectivas classes do módulo de apresentação. Para adicionar um novo critério basta criar uma nova classe que estenda a classe *ConceitoCT* e implemente o método *avaliaCódigo.* Por herança esse novo conceito, receberá a instância de código, com todas as estatísticas e estruturas de dados de todos os códigos de telas do projeto App Inventor. A classe *AppInventorGrader* é o controle do analisador, ela deve obter a pontuação do novo conceito, colocar a pontuação na instância de *AppInventorGrade* e somar na variável de pontuação total.

Adicionando novo analisador e avaliador de código.

Adicionar um novo analisador e avaliador de código diz respeito a incluir no CodeMaster a análise e avaliação de projetos de uma nova linguagem de programação. O núcleo do novo analisador e avaliador deve ser implementado, isso é composto por reconhecimento de projeto, extração de código, análise e estatística dos componentes e avaliação de critérios.

Primeiramente para adicionar um novo analisador ao sistema, deve-se criar no módulo de apresentação: telas de resultado para o novo analisador, classe *Grade* do novo analisador que receberá o resultado da avaliação, além de adicionar a nova opção de linguagem na tela de avaliação do professor e novos critérios se os disponíveis não forem suficientes na classe *ProjectSettings.* Na classe *Upload* deve ser liberado o upload de arquivos na extensão de projeto do novo analisador.

No módulo de avaliação e análise toda a implementação do novo analisador deve ser inserida, basta que a classe principal do novo analisador seja estendida da classe *Grader* e que o método *gradeProject* seja implementado. Esse método recebe o *ProjectSettings* com os critérios que devem ser analisados, o nome do projeto e o arquivo de projeto em si. Este método retorna uma instância de *Grade* que contém as pontuações de cada critério analisado no novo analisador, a pontuação total, a nota de 0 a 10 e o nível ninja. A classe *RESTGrader* é a responsável por direcionar o fluxo de execução para o analisador correspondente do projeto recebido, portanto, nela deve-se permitir o novo tipo de projeto que virá especificado no *ProjectSettings* adicionando uma nova condicional de tipo de projeto e instanciando o novo analisador.

4.4.5 DESIGN DE INTERFACE DO CODEMASTER

O design de interface do CodeMaster foi feito de forma iterativa, para cada fluxo de execução proposto foi desenvolvida a interface gráfica do sistema. Com a intenção de criar uma interface gráfica com boa usabilidade seguindo padrões de identidade visual definidos pela iniciativa Computação na Escola, a interface gráfica foi elaborada em conjunto com os bolsistas Heliziane Barbosa e Luiz Felipe Azevedo da iniciativa Computação na Escola do GQS/InCod/INE/UFSC.

Para cada caso de uso proposto é apresentado abaixo o design de interface criada bem como o fluxo de execução principal.

Tabela 22 - Interfaces gráficas do USC01

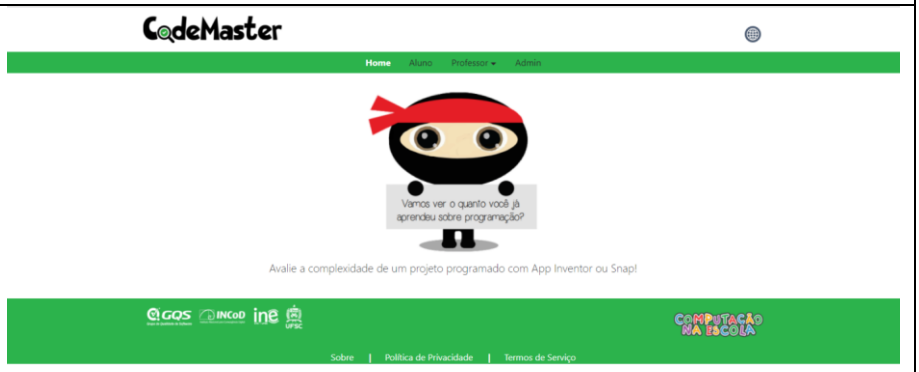
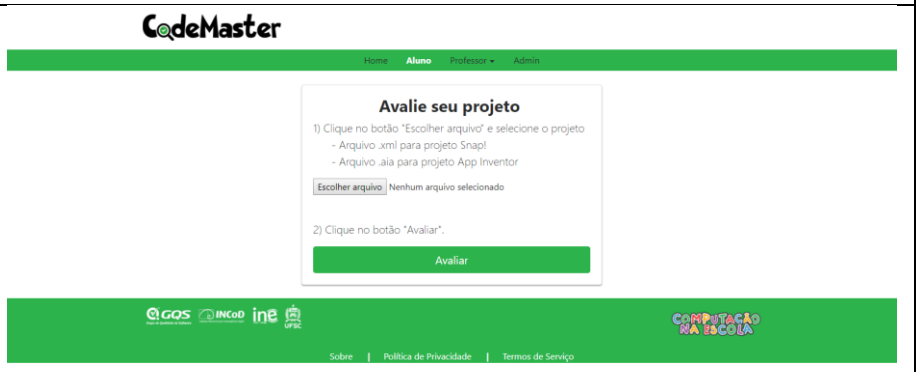
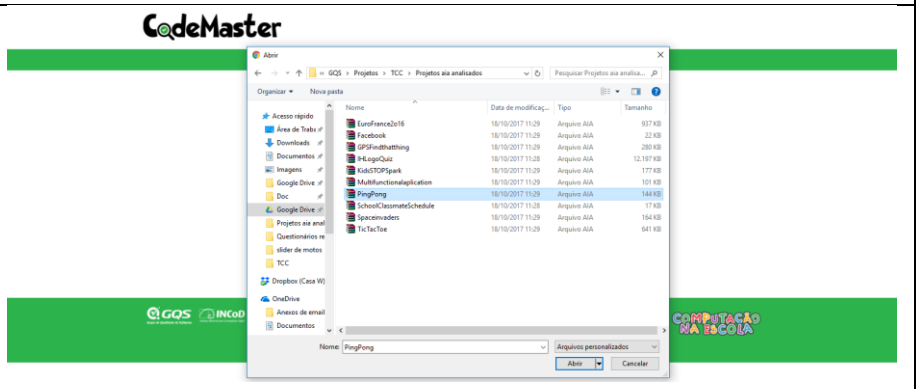
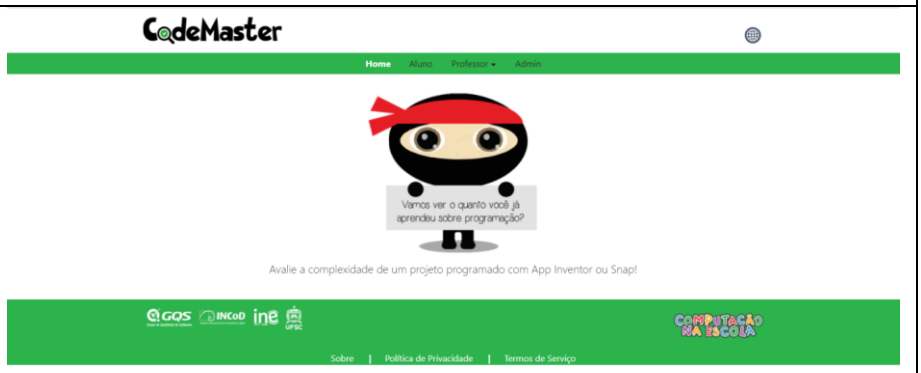
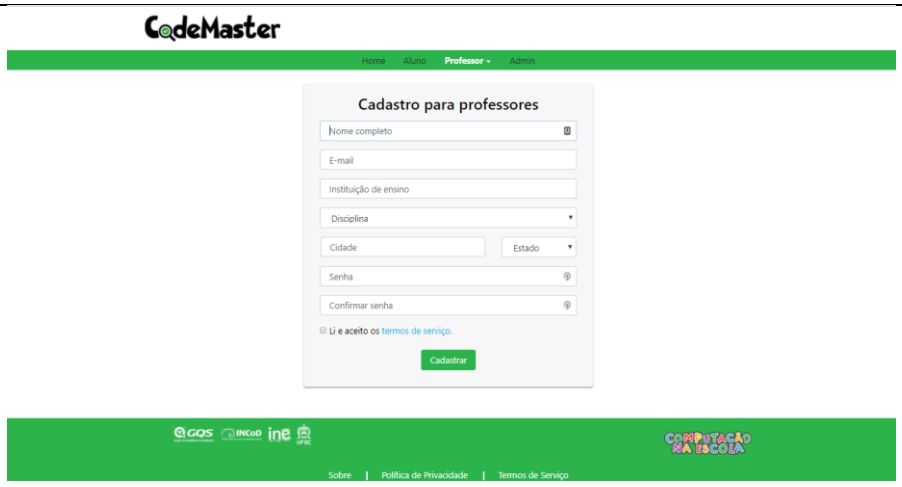
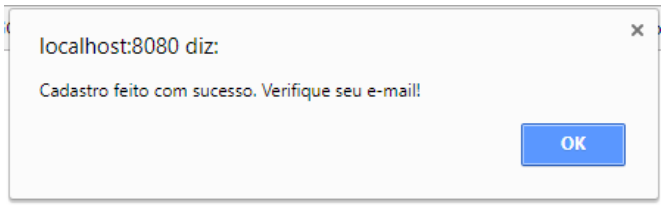
USC01 - Análise Código Pelo Aluno	
1. Aluno acessa o site do CodeMaster.	
2. Aluno clica no item de menu “Aluno”.	
3. Aluno clica em escolher arquivo e escolhe projeto em seu computador.	
4. Aluno clica em “Avaliar” e sistema retorna o resultado da análise do projeto.	

Tabela 23 - Interfaces gráficas do USC02

USC02 - Cadastro Professor	
1. Professor acessa site do CodeMaster.	
2. Professor clica no menu superior no botão “professor”.	
3. Professor clica na opção de “Ainda não sou cadastrado” abaixo dos campos de login e senha.	
4. Professor informa, nome, e-mail, senha, instituição de ensino, disciplina, cidade, estado e clica em cadastrar.	

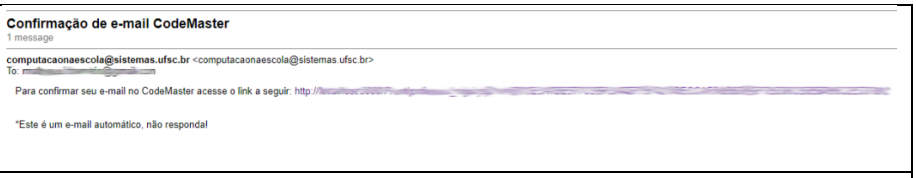
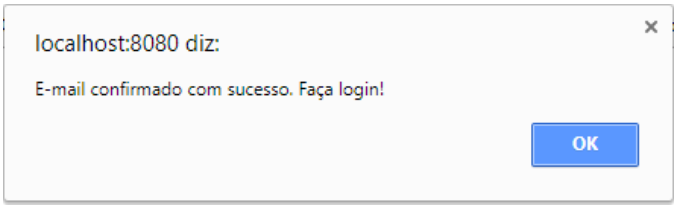
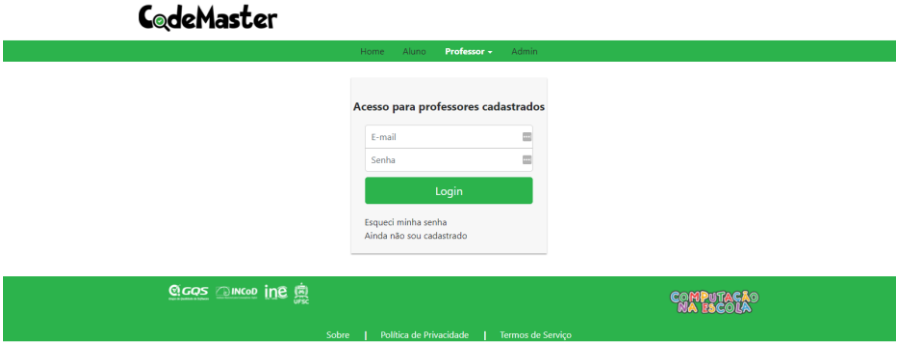
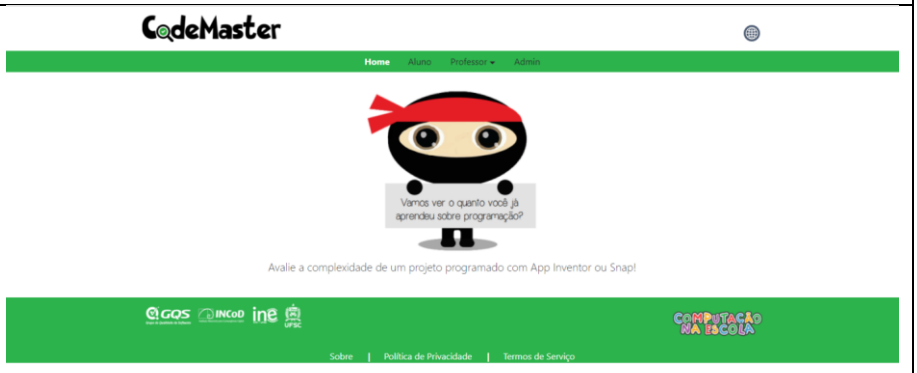
5. Professor recebe e-mail com link para validar conta.	
6. Professor abre link recebido no e-mail e uma mensagem de conta confirmada é exibida.	
7. Página da área de login de professor é exibida.	

Tabela 24 - Interfaces gráficas do USC03

USC03 - Login de Professor	
1. Professor acessa o site do CodeMaster.	
2. Professor clica no botão professores no menu superior da tela.	

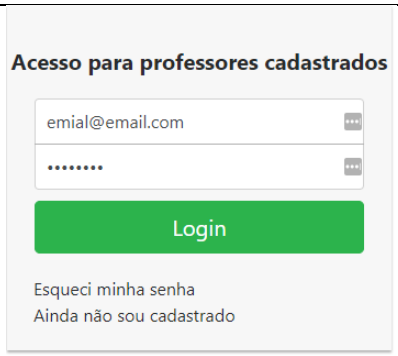
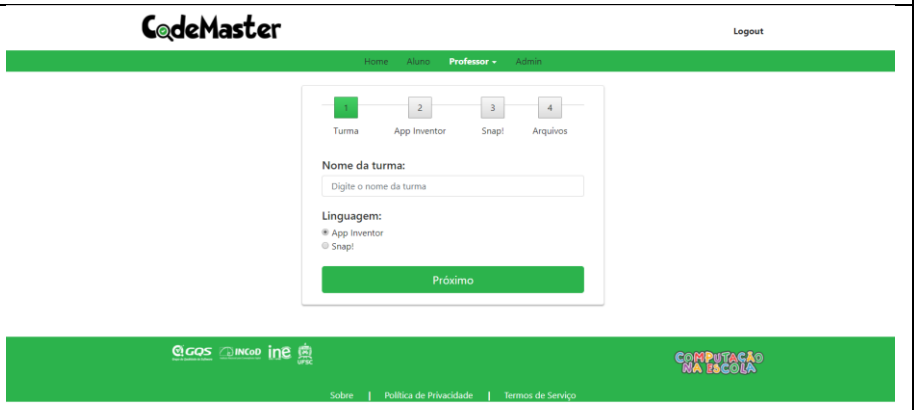
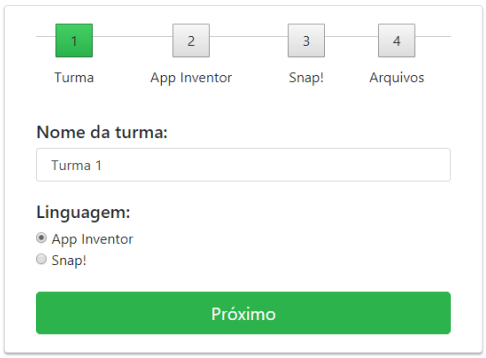
3. Professor informa seu e-mail e senha nos campos de login e senha e clica em Login.	
4. O sistema autentica o professor e é redirecionado à página principal do modo professor.	

Tabela 25 - Interfaces gráficas do USC04

USC04 – Análise de Múltiplos Projetos	
1. Professor digita o nome da turma e seleciona o tipo de projeto que deseja avaliar e clica em próximo.	

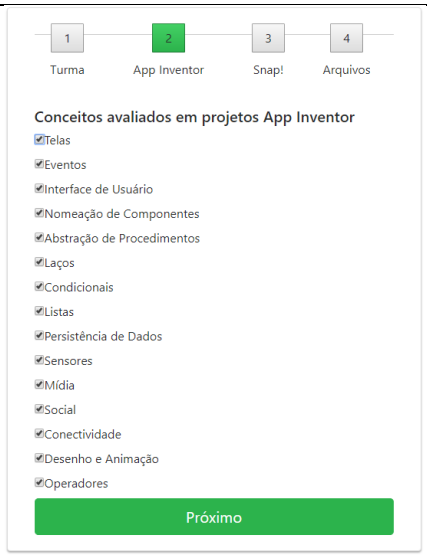
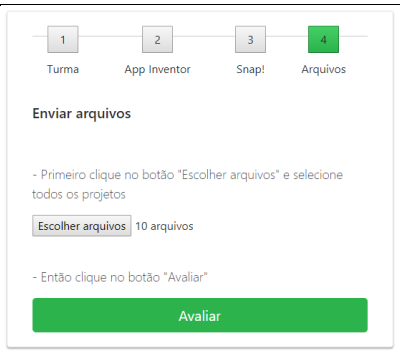
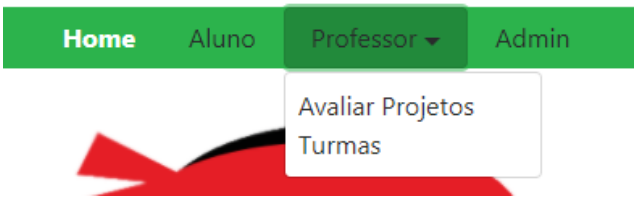
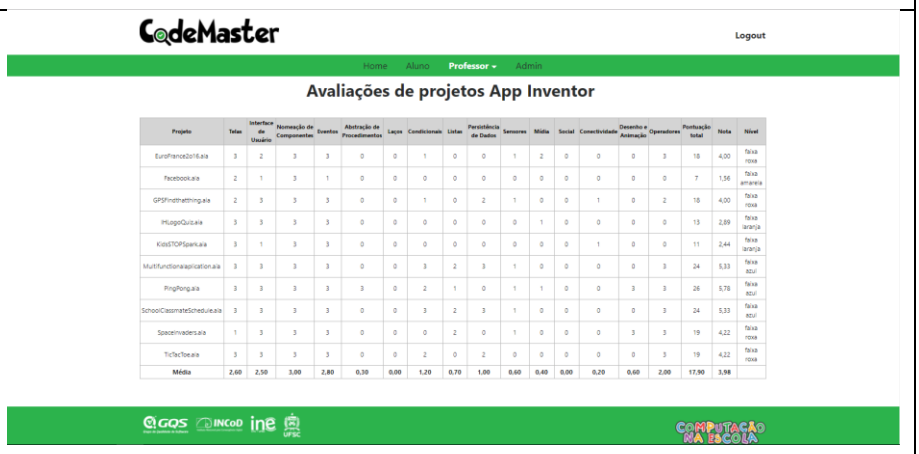
2. Professor seleciona os conceitos que deseja avaliar.	
3. Professor escolhe múltiplos arquivos de seu computador que correspondem a uma turma.	
4. Professor clica em avaliar e recebe o resultado da análise de todos os projetos em uma tabela.	

Tabela 26 - Interfaces gráficas do USC05

USC05 - Ver Projetos Analisados	
1. Professor clica em “Turmas” no menu superior da área de professores.	
2. Professor seleciona a turma que deseja ver as análises.	
3. Sistema consulta no Banco de Dados e retorna o resultado das análises da turma selecionada.	

A ferramenta CodeMaster v1.0 está disponível online em: http://apps.computacaonaescola.ufsc.br:8080.

5. AVALIAÇÃO DO CODEMASTER – APP INVENTOR

Com o objetivo de avaliar a qualidade do CodeMaster – App Inventor foram realizadas duas avaliações, uma voltada a avaliação do CodeMaster – App Inventor em relação a utilidade, funcionalidade, eficiência e usabilidade pela perspectiva de professores e alunos no contexto do ensino de computação e outra voltada a avaliação da corretude da ferramenta.

Desta forma foram realizadas as seguintes avaliações:

- Um teste de corretude comparando o resultado da avaliação automática gerada pelo CodeMaster – App Inventor e da avaliação manual de códigos; e
- Uma avaliação de usuários de forma a obter dados sobre a eficácia em relação as tarefas propostas, bem como sobre a qualidade percebida pelo ponto de vista de professores e alunos.

5.2 AVALIAÇÃO DA CORRETUDE

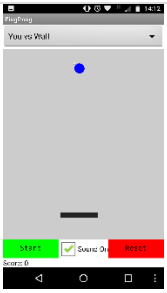
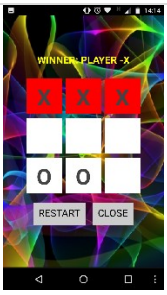
Nesta avaliação, o objetivo é avaliar a corretude das avaliações de projetos App Inventor executadas pelo CodeMaster. Para isso são comparadas avaliações de projetos feitas manualmente, usando a rubrica da seção 4.1, com os resultados das avaliações feitas automaticamente pelo CodeMaster.

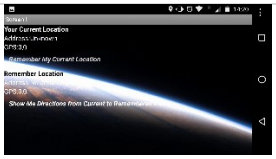
Espera-se que utilizando ambas as maneiras os resultados sejam os mesmos, demonstrando a corretude da avaliação.

5.2.1 EXECUÇÃO DA AVALIAÇÃO DA CORRETUDE

Para executar a avaliação da corretude, foram selecionados aleatoriamente 10 projetos da seção de aplicativos populares da galeria de aplicativos do App Inventor. Os projetos têm diferentes propostas, a maior parte deles são propostas de jogos, mas cada um com lógicas diferentes, como um jogo de Ping Pong e outro de Quiz. Os projetos selecionados são apresentados na Tabela 27.

Tabela 27 - Projetos App Inventor selecionados para avaliação da corretude

Nome	Disponível em	Captura de tela
Multifuncional Aplication	ai2.appinventor.mit.edu/?galleryId=4618340131930112	
PingPong	ai2.appinventor.mit.edu/?galleryId=4833713599283200	
TicTacToe	ai2.appinventor.mit.edu/?galleryId=5710777644744704	
Kids, STOP! - Spark!	ai2.appinventor.mit.edu/?galleryId=4720218625540096	
EuroFrance2o16	ai2.appinventor.mit.edu/?galleryId=5405102169128960	
Space_invaders	ai2.appinventor.mit.edu/?galleryId=5918483228393472	

Facebook	ai2.appinventor.mit.edu/?galleryId=5142515749748736	
GPS - Find that thing!	ai2.appinventor.mit.edu/?galleryId=5136102976389120	
School Classmate&Schedule	ai2.appinventor.mit.edu/?galleryId=5407619303604224	
IH Logo Quiz	ai2.appinventor.mit.edu/?galleryId=4776617318023168	

Para a avaliação manual dos códigos, cada projeto selecionado foi aberto no próprio App Inventor e para cada conceito da rubrica foi verificada e contada a presença dos blocos e componentes visuais. Analisando os aplicativos manualmente se obteve as pontuações, notas e níveis mostradas na Figura 21.

Projeto	Telas	Interface de usuário	Nomeação variáveis e procedimentos	Eventos	Abstração de procedimentos	Laços	Condicionais	Listas	Persistência dados	Sensores	Mídia	Social	Conectividade	Desenho e animação	Operadores	Pontuação Total	Nota	Nível
IHLogoQuis	3	3	3	3	0	0	0	0	0	0	1	0	0	0	0	13	2,89	Laranja
Multifuncional Aplication	3	3	3	3	0	0	3	2	3	1	0	0	0	0	3	24	5,33	Azul
KidsSTOP Spark	3	1	3	3	0	0	0	0	0	0	0	0	1	0	0	11	2,44	Laranja
School Classmate Schedule	3	3	3	3	0	0	3	2	3	1	0	0	0	0	3	24	5,33	Azul
EuroFrance 2016	3	2	3	3	0	0	1	0	0	1	2	0	0	0	3	18	4	Roxa
PingPong	3	3	3	3	3	0	2	1	0	1	1	0	0	3	3	26	5,78	Azul
GPS find thing	2	3	3	3	0	0	1	0	2	1	0	0	1	0	2	18	4	Roxa
Space Invaders	1	3	3	3	0	0	0	2	0	1	0	0	0	3	3	19	4,22	Roxa
Facebook	2	1	3	1	0	0	0	0	0	0	0	0	0	0	0	7	1,56	Amarela
ticTacToe	3	3	3	3	0	0	2	0	2	0	0	0	0	0	3	19	4,22	Roxa
Média	2,6	2,5	3	2,8	0,3	0	1,2	0,7	1	0,6	0,4	0	0,2	0,6	2	17,9	3,98	

Figura 21 - Resultados da avaliação manual de projetos App Inventor

Após a avaliação manual, os 10 projetos foram submetidos à avaliação pelo CodeMaster no modo professor. O resultado obtido é apresentado na Figura 22.

Projeto	Telas	Interface de Usuário	Nomeação de Componentes	Eventos	Abstração de Procedimentos	Laços	Condicionais	Listas	Persistência de Dados	Sensores	Mídia	Social	Conectividade	Desenho e Animação	Operadores	Pontuação total	Nota	Nível
IHLogoQuiz.aia	3	3	3	3	0	0	0	0	0	0	1	0	0	0	0	13	2,89	faixa laranja
Multifunctionalaplication.aia	3	3	3	3	0	0	3	2	3	1	0	0	0	0	3	24	5,33	faixa azul
KidsSTOPSpark.aia	3	1	3	3	0	0	0	0	0	0	0	0	1	0	0	11	2,44	faixa laranja
SchoolClassmateSchedule.aia	3	3	3	3	0	0	3	2	3	1	0	0	0	0	3	24	5,33	faixa azul
EuroFrance2o16.aia	3	2	3	3	0	0	1	0	0	1	2	0	0	0	3	18	4,00	faixa roxa
PingPong.aia	3	3	3	3	3	0	2	1	0	1	1	0	0	3	3	26	5,78	faixa azul
GPSFindthatthing.aia	2	3	3	3	0	0	1	0	2	1	0	0	1	0	2	18	4,00	faixa roxa
SpaceInvaders.aia	1	3	3	3	0	0	0	2	0	1	0	0	0	3	3	19	4,22	faixa roxa
Facebook.aia	2	1	3	1	0	0	0	0	0	0	0	0	0	0	0	7	1,56	faixa amarela
TicTacToe.aia	3	3	3	3	0	0	2	0	2	0	0	0	0	0	3	19	4,22	faixa roxa
Média	**2,60**	**2,50**	**3,00**	**2,80**	**0,30**	**0,00**	**1,20**	**0,70**	**1,00**	**0,60**	**0,40**	**0,00**	**0,20**	**0,60**	**2,00**	**17,90**	**3,98**	

Figura 22 - Avaliação pelo CodeMaster de Projetos App Inventor

Com esses resultados, é possível verificar a corretude do CodeMaster para todos os projetos selecionados. Todos obtiveram as pontuações esperadas para cada conceito analisado. Também, é possível verificar a corretude das fórmulas de cálculo da nota final e nível ninja. Além da avaliação formal com os 10 projetos citados, ao longo do desenvolvimento do CodeMaster muitos testes foram executados com projetos App Inventor diferentes e os resultados também se mostraram consistentes com o definido na rubrica apresentada.

5.3 AVALIAÇÃO POR USUÁRIOS

5.3.1 DEFINIÇÃO DA AVALIAÇÃO POR USUÁRIOS

A avaliação do usuário é, basicamente, uma forma sistemática de observar o usuário testando uma ferramenta e coletar informações específicas que permite determinar se a ferramenta é fácil ou difícil para eles. Assim são realizados testes de usabilidade para coletar os dados.

O objetivo desta avaliação é analisar a qualidade do CodeMaster em termos de utilidade, funcionalidade, desempenho e usabilidade pelo ponto de vista de professores e estudantes. Os fatores de qualidade avaliados são apresentados na Tabela 28.

Tabela 28 - Fatores de qualidade analisados

Característica	**Sub-característica**	**Avaliação do usuário**		
		Questionário		**Observação**
		Questionário de professor	**Questionário de aluno**	
Utilidade		Você acha a ferramenta CodeMaster útil no ensino de computação no ensino Básico?	Você acha a ferramenta CodeMaster útil na aprendizagem de programação?	
		Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) é prático no seu dia-a-dia?		
Funcionalidade	Completude	Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta?		
		Você acha que existem aspectos relevantes no (processo de) avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta?		
		Você acha que as informações disponibilizadas como resultado da avaliação são suficientes?	Você acha que as informações disponibilizadas como resultado da avaliação são suficientes?	
	Corretude	Você observou algum erro (bug) em relação a funcionalidade da ferramenta?	Você observou algum erro (bug) em relação a funcionalidade da ferramenta?	
			Você achou a nota justa?	
Desempenho	Tempo de resposta	A performance (tempo de resposta) do sistema é satisfatória?	A performance (tempo de resposta) do sistema é satisfatória?	
Usabilidade	Efetividade			Usuário completou a tarefa
	Satisfação	Eu penso que usarei este sistema com frequência.	Eu penso que usarei este sistema com frequência.	

		Acho o sistema desnecessariamente complexo.	Acho o sistema desnecessariamente complexo.	
		Penso que o sistema é fácil de usar.	Penso que o sistema é fácil de usar.	
		Acho que vou precisar da ajuda de um técnico para usar este sistema.	Acho que vou precisar da ajuda de um técnico para usar este sistema.	
		Acho as funções deste sistema bem integradas.	Acho as funções deste sistema bem integradas.	
		Encontro muitas inconsistências neste sistema.	Encontro muitas inconsistências neste sistema.	
		Imagino que as pessoas aprenderão rapidamente a usar este sistema.	Imagino que as pessoas aprenderão rapidamente a usar este sistema.	
		Não acho o sistema prático de usar.	Não acho o sistema prático de usar.	
		Senti-me confiante ao usar o sistema.	Senti-me confiante ao usar o sistema.	
		Precisei aprender muitas coisas antes de ser capaz de operar o sistema.	Precisei aprender muitas coisas antes de ser capaz de operar o sistema.	
	Operabilidade	Você achou fácil de usar o sistema?	Você achou fácil de usar o sistema?	
		Você acha que a ferramenta possui elementos ambíguos ou difíceis de entender?	Você acha que a ferramenta possui elementos ambíguos ou difíceis de entender?	

Os dados foram coletados por meio de observações e um questionário pós teste. O questionário é definido por meio dos fatores sendo avaliados apresentados na Tabela 28. Os questionários utilizados são apresentados nos Anexo A e B.

No teste de usabilidade realizado, os usuários, primeiramente, recebem uma visão geral do objetivo do CodeMaster e devem executar uma tarefa pré-definida (avaliar um ou um conjunto de projetos de programação do App Inventor com o CodeMaster). A descrição detalhada da tarefa é apresentada nos Anexo A e B.

5.3.2 EXECUÇÃO DA AVALIAÇÃO POR USUÁRIOS

A execução da avaliação da ferramenta foi realizada por professores e alunos ligados a iniciativa Computação na Escola. É pressuposto que todos os participantes tivessem conhecimento básico sobre o que é, e sobre as funcionalidades do App Inventor.

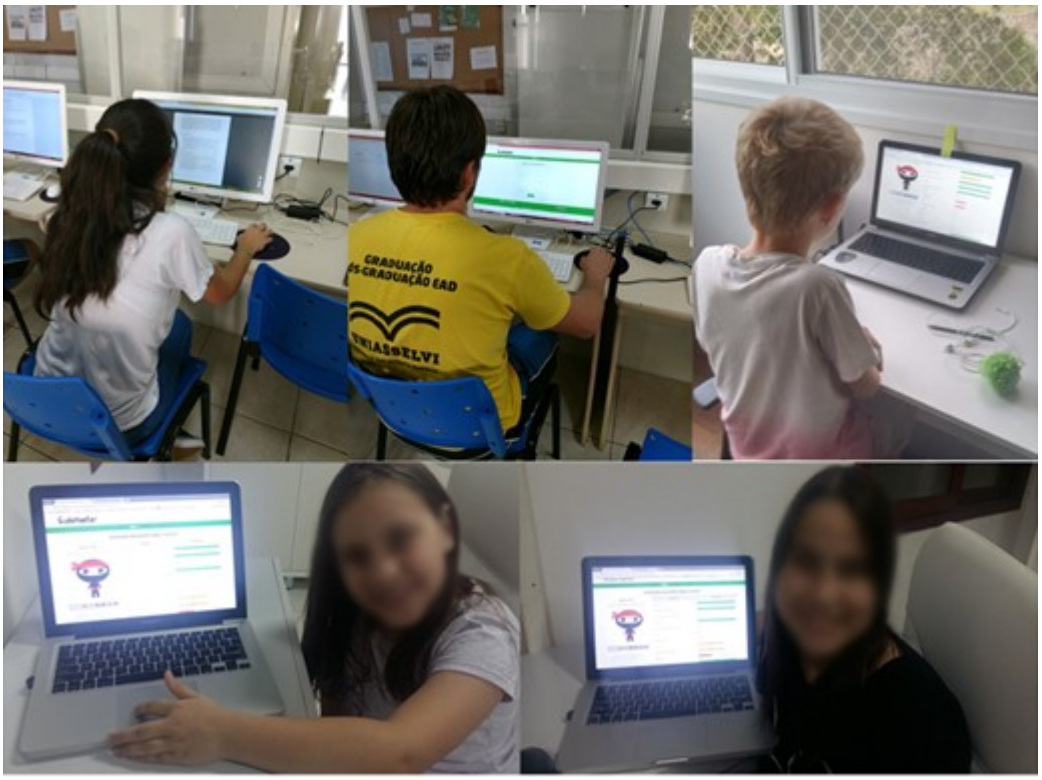


Figura 23 - Fotos de alguns avaliadores do CodeMaster

No total foram realizadas 8 avaliações da ferramenta, sendo 4 avaliações de alunos utilizando o modo aluno do CodeMaster e 4 avaliações de professores ou jovens tutores (jovens alunos com conhecimento mais avançado sobre programação que ajudam no ensino de computação para outros jovens) que avaliaram o modo de professor da ferramenta. A avaliação ocorreu como planejada no período dos meses de setembro e outubro de 2017 e todos os avaliadores responderam o questionário ao final da utilização do CodeMaster.

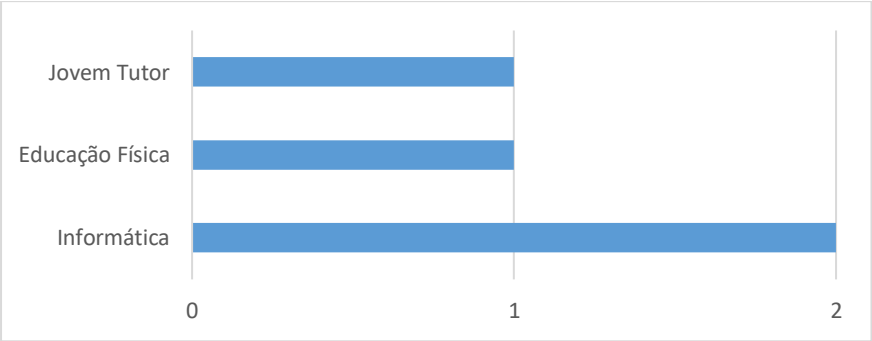


Figura 24 - Quantidade de professores por disciplina

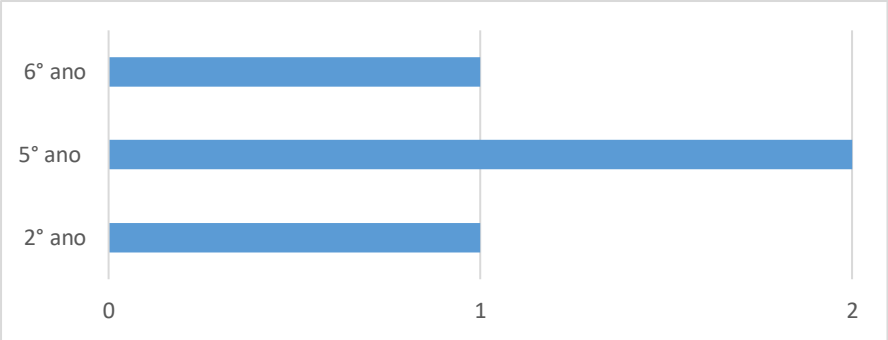


Figura 25 - Quantidade de alunos por ano do ensino

5.3.3 ANÁLISE DOS DADOS

Para uma melhor interpretação das informações, as respostas são agrupadas por critério de análise, apresentando a quantidade referente a respostas afirmativas e negativas.

O CodeMaster é útil?

Todos os usuários consideram o CodeMaster útil para a aprendizagem e ensino de programação. De forma geral conceitos como o entendimento de programação, organização em avaliações e diferenciação das partes de um aplicativo foram elogiados do CodeMaster. A forma de fazer o upload de projetos foi considerada positiva, porém algumas sugestões foram dadas, como por exemplo, a possibilidade do próprio aluno fazer o envio do projeto para o professor via o sistema. Os dados coletados sobre a utilidade são apresentados na tabela 29.

Tabela 29 - Análise da Utilidade do CodeMaster por alunos

Questão de análise	**Total de Respostas**	
	Sim	**Não**
Você acha a ferramenta CodeMaster útil na aprendizagem de programação?	5 alunos	0 alunos
Você acha a ferramenta CodeMaster útil no ensino de computação no ensino Básico?	3 professores	0 professores
Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) é prático no seu dia-a-dia?	2 professores	1 professores
Comentários		
	“Porque mostra se os alunos estão indo bem ou não.”	

Você acha a ferramenta CodeMaster útil na aprendizagem de programação? Explique, porque?	“Porque você pode criar um programa (no Snap! ou no App Inventor), e ser avaliado para ver se seu jogo/app está bom ou se precisa de algumas melhorias.”
Você acha a ferramenta CodeMaster útil no ensino de computação no ensino Básico? Explique, porque?	“Penso que é uma forma rápida de avaliar os alunos e ainda verificar a profundidade do entendimento que eles obtiveram.”
	“Eu acho útil o CodeMaster, porque podemos nos organizar melhor com as avaliações.”
	“Vejo como importante principalmente na conceituação e diferenciação das diferentes partes de um aplicativo como a conectividade, telas, mídias...”
Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) é prático no seu dia-a-dia? Explique, porque?	“Uma forma em que cada aluno submetesse seu próprio app para posterior verificação completa e de toda turma por parte do professor também seria útil, pois nem sempre o professor fica com uma cópia do projeto feito pelo aluno.”
	“Mas penso que o nome pode ser o do app, para facilitar e já dar uma ideia do que se trata o app.”

O CodeMaster é funcional?

Em relação a funcionalidade, a ferramenta obteve um feedback positivo, foi observado um erro durante a execução da análise por um dos avaliadores que deve ser corrigido.

Além disso a nota obtida pelos alunos, em sua maioria achou justa, porém, foi comentado em uma das avaliações em que mesmo o projeto analisado recebendo *upgrades* a nota não aumentava como o aluno esperava. Isso pode ser um fator que desanima o aluno no seu processo de aprendizagem, por isso, futuramente é importante verificar especificamente a necessidade de uma atualização na forma de cálculo da nota final.

Fazendo análise da Tabela 30, observa-se que a completude das funcionalidades da ferramenta tem um *feedback* positivo, porém, em comentários, foi sugerida a inserção de mais alguns critérios na análise de código, como paralelismo, duplicidade de código e presença de código morto. Também foi comentado que seria interessante ser possível ordenar a listagem de projetos analisados por um professor por cada uma das colunas presentes no resultado da avaliação que é apresentada.

Tabela 30 - Análise da Funcionalidade do CodeMaster por alunos

Questão de análise	**Total de Respostas**	
	Sim	**Não**
Você acha que as informações disponibilizadas como resultado da avaliação são suficientes?	5 alunos 2 professores	0 alunos 1 professor
Você acha que existem aspectos relevantes no (processo de) avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta?	1 professor	2 professores
Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta?	1 professor	2 professores
Você observou algum erro (bug) em relação a funcionalidade da ferramenta?	1 aluno 1 professor	4 alunos 2 professores
Você achou a nota justa?	4 alunos	1 aluno
Comentários		
Você acha que as informações disponibilizadas como resultado da avaliação são suficientes? Se não, quais devem ser adicionados?	“Seria útil a inclusão de ordenação por cada um dos itens, telas, interfaces... e também pontuação total, nota e faixa.”	
	“Seria interessante aparecer uma imagem no nível, ao invés da escrita faixa branca, ou ambos, ou a escrita com a cor da faixa.”	
Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta? Se sim, quais?	“Paralelismo; código que não é usado, morto; trechos de código duplicados.”	
Você achou a nota justa? Se não, porque?	“O meu projeto foi um “update” do projeto de demonstração e tive uma nota pior.”	

Como é o desempenho do CodeMaster?

Na análise do desempenho apresentado na tabela 31, a ferramenta recebeu um feedback relativo, sabe-se que o fator determinante para o desempenho é a velocidade e qualidade de conexão de internet do usuário, como cada projeto tem que ser enviado ao CodeMaster para análise, projetos maiores podem ter um tempo de

resposta maior também. Percebe-se por meio dos dados da Tabela 31 que os alunos têm menos tolerância quanto a velocidade da ferramenta, já por parte dos professores não houve pontos negativos.

Tabela 31 - Análise de desempenho do CodeMaster

Questão de análise	**Total de Respostas**	
	Sim	**Não**
A performance (tempo de resposta) do sistema é satisfatória?	3 alunos 3 professores	2 alunos 0 professores

Como é a usabilidade do CodeMaster?

Em geral a ferramenta obteve um feedback muito positivo na facilidade de uso perante os alunos e professores, observou-se apenas dificuldade de compreender termos ligados diretamente a conceitos de programação. Na Tabela 32 são apresentados os dados da análise da facilidade de uso. Os avaliadores confirmam que o sistema é fácil de usar, porém com alguns detalhes que podem ser melhorados posteriormente, como breves explicações dos conceitos apresentados no resultado da avaliação de projetos.

Tabela 32 - Análise da facilidade de uso do CodeMaster

Questão de análise	**Total de Respostas**	
	Sim	**Não**
Você acha que a ferramenta possui elementos ambíguos ou difíceis de entender?	1 aluno 1 professor	4 alunos 2 professores
Você achou fácil de usar o sistema?	5 alunos 3 professores	0 alunos 0 professores
Comentários		
Você acha que a ferramenta possui elementos ambíguos ou difíceis de entender? Se sim, quais?	“Por exemplo: eu não sei o que é laço em linguagem de computação.”	
	“Especialmente para quem não é da área poderia ter uma explicação rápida ao passar o mouse em cima dos itens avaliados, quando aparece a tabela de notas. Os mais técnicos	

	também poderiam explicar, rapidamente o conceito.”

A avaliação de usabilidade em termos de satisfação foi feita por meio de um diagrama de SUS e observou-se uma pontuação média de 89,68 pontos de usabilidade. No requisito não funcional 5 foi definido um mínimo de 80 pontos de usabilidade, portanto a pontuação obtida é satisfatória. Além disso, todos os avaliadores conseguiram concluir a tarefa.

Tabela 33 - Pontuações de Usabilidade do CodeMaster

Pontuações SUS					**Média**
Alunos:	77,5	97,5	95	95	91,25
Professores:	95	70	87,5	100	88,125
				Total média:	89,68

Pontos fortes

Com base no feedback dos usuários o CodeMaster - App inventor é considerado útil, funcional, eficiente em desempenho e com boa usabilidade dentro do esperado para a ferramenta. Os pontos fortes percebidos pelos alunos avaliadores são que a ferramenta tem uma linguagem adequada para a idade dos jovens. A ideia de usar um ninja para apresentar o nível de programação foi bastante elogiada. Os alunos mostraram empolgação na hora que estavam programando seus aplicativos, ficaram entusiasmados com a possibilidade de melhorar seus códigos e por consequência melhorar as notas obtidas no CodeMaster.

Por parte dos professores o principal ponto forte é a possibilidade de ter uma ferramenta que dá apoio a tomada de decisão no momento de avaliar os códigos dos alunos e pela organização que os critérios de avaliação são expostos, dando uma maior noção dos diferentes elementos que foram utilizados no código.

Sugestões de melhoria

As sugestões de melhoria são principalmente relacionadas a adição de novos critérios de avaliação ao CodeMaster, como a análise de paralelismo de código, presença de código duplicado e presença de código morto (sem utilidade). Outra sugestão é para o sistema de upload de múltiplos projetos no modo professor, foi sugerida a melhoria da função, permitindo que os próprios alunos enviem os seus projetos ao professor pelo sistema e/ou de alguma maneira poder renomear os nomes de alunos e projetos na hora do envio. Além disso foi sugerida a inclusão da funcionalidade de o professor poder ordenar os projetos de uma turma de alunos por cada um dos critérios avaliados, nota final e nível ninja.

Ameaças a validade da avaliação

Vale ressaltar que durante período de aplicação dos testes aos alunos e professores o CodeMaster ainda se encontrava em desenvolvimento, portanto a avaliação foi feita numa primeira versão que poderia retornar erros e inconsistências durante o processo de avaliação de código. Outra ameaça a validade da avaliação feita é a quantidade de usuários capacitados disponíveis para avaliar a ferramenta, entende-se que mais usuários testando e avaliando poderia retornar resultados com mais precisão e confiabilidade.

6. CONCLUSÃO

O objetivo principal deste trabalho foi desenvolver uma ferramenta web para análise e avaliação de código gerado pelo App Inventor para dar suporte ao ensino de computação ao Ensino Básico. Neste contexto, foi feita a análise da fundamentação teórica (O1). Também foi realizada a análise do estado da arte identificando a falta de ferramentas de análise de código para suporte ao ensino de computação de modo generalizado (O2). Deste modo, baseando-se na fundamentação teórica e na análise do estado da arte, foi elaborado um modelo conceitual incluindo a definição de uma rubrica para análise de código do App Inventor e foi desenvolvida uma ferramenta que analisa e avalia automaticamente projetos de apps criados com o App Inventor (O3). O desenvolvimento da ferramenta CodeMaster - App Inventor foi feito de forma integrada e em conjunto com Pelle (2018) que implementou o CodeMaster – Snap!. Os resultados da avaliação do CodeMaster - App Inventor fornecem uma indicação inicial da sua corretude, utilidade, funcionalidade, desempenho e usabilidade muito positiva (O4).

O CodeMaster permite o cadastro de professores e a análise e avaliação de múltiplos projetos em lote, podendo ser agrupados por turmas de alunos. Além do serviço disponibilizado a professores que querem ensinar programação para seus alunos, a ferramenta permite que alunos possam avaliar seus projetos e receberem um *feedback* de como está o seu nível de competência em programação por meio de notas e faixas de ninja que tornam o sistema mais amigável para os jovens aprendizes. Por falta de tempo hábil, a parte de pesquisador, que deve exibir estatísticas gerais de projetos analisados, citada nos requisitos funcionais, não foi implementada e não está disponível na primeira versão do CodeMaster.

Com o CodeMaster – App Inventor, existe hoje uma ferramenta robusta e única para análise e avaliação automatizada de códigos produzidos no ambiente de programação visual App Inventor, dando suporte a professores e alunos no ensino e aprendizagem de computação no Ensino Básico.

Como trabalhos futuros, recomenda-se a implementação de análise de usabilidade dos projetos App Inventor, por exemplo, analisando a posição de elementos gráficos dos aplicativos, ou cores utilizadas. Além disso, recomenda-se a pesquisa da necessidade de implementação dos critérios de paralelismo, trechos de código duplicados e trechos de código morto além de sua implementação caso se faça

necessário. Outro trabalho que pode ser desenvolvido é a integração do módulo de análise e avaliação do CodeMaster diretamente com o App Inventor, por meio da implementação de uma extensão que permita avaliar o código sem precisar sair do ambiente de programação do próprio App Inventor. Recomenda-se também, a continuação da implementação do modo de pesquisador do CodeMaster que tem como objetivo apresentar estatísticas gerais de todas as análises feitas com a ferramenta.

REFERÊNCIAS

AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compilers, Principles, Techniques**. Boston: Addison Wesley, 1986.

APP INVENTOR, 2017. Disponível em: <http://appinventor.mit.edu/explore/>. Acesso em: junho de 2017.

AUTONOMIA, 2017 Disponível em: <http://autonomia.com.br/fundamental-2-perguntas-e-respostas/>. Acesso em: junho de 2017.

BALL, M. A.; GARCIA, D. D. **Autograding and Feedback for Snap!: A Visual Programming Language.** In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education. Memphis, TN, USA, 2016.

BARANOV, S. P. et al. **Pedagogía.** La Habana: Pueblo y Educación, 1989.

BARBOSA, A F. **Pesquisa sobre o uso da internet por crianças e adolescentes no Brasil.** TIC kids online brasil 2014. Brasil, 2015.

BILAL, M., CHAN, P., MEDDINGS, F., & KONSTADOPOULOU, A. **SCORE: An advanced assessment and feedback framework with a universal marking scheme in higher education.** In Proc. of the Int. Conf. on Education and e-Learning Innovations, Sousse/Tunísia, 2012, 1-6.

BLACK, P., & WILIAN, D. **Assessment and classroom learning**. Assessment in Education: Principles, Policy & Practice, 5(1), 1998.

BLOOM, B. S. **Taxonomy of Educational Objectives, The Classification of Educational Goals**. New York: David McKay, 1956.

BLOCKLY, 2017. Disponível em: <https://developers.google.com/blockly/>. Acesso em: setembro de 2017.

BOE, B. et al. **Hairball: Lint-inspired static analysis of scratch projects.** In: Proceeding of the 44th ACM technical symposium on Computer science education. Colorado, USA, 2013.

BRANCH, R. M. **Instructional Design**: The ADDIE Approach. New York, 2009.

BRASIL. **Lei de Diretrizes e Bases da Educação Nacional** - Lei Nº 9.394. Brasília. 1996.

BRENNAN, K., RESNICK, M. **New frameworks for studying and assessing the development of computational thinking.** Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada, 2012.

CNE. GQS/INCoD/INE/UFSC. **Iniciativa Computação na Escola**, 2013. Disponível em: <http://www.computacaonaescola.ufsc.br>. Acesso em: setembro de 2016.

CLARK, D. R. **ADDIE Model**, 2009. Disponível em: <http://www.nwlink.com/~donclark/hrd/bloom.html>. Acesso em: setembro de 2016.

CODE. Code.org, 2013. Disponível em: <https://code.org/>. Acesso em: setembro 2016.

CORTESÃO, L. **Formas de ensinar, formas de avaliar: breve análise de práticas correntes de avaliação**. Reorganização curricular do ensino básico: avaliação das aprendizagens: das concepções às novas práticas, Brasil, 2002.

CSTA. ACM. **CSTA K –12 Computer Science Standards**, 2011. Disponível em: <http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf>. Acesso em: setembro de 2016.

Daniel, G. T., von Wangenheim, C. G., Araújo, G., de Medeiros, S., & da Cruz Alves, N. **Ensinando a Computação por meio de Programação com App Inventor**. Anais do Computer on the Beach, Florianópolis, Brasil, 2017, 357-365.

ESERYEL, D., IFENTHALER, D., XUN, G. **Validation study of a method for assessing complex ill-structured problem solving by using causal representations**. Educational Technology Research and Development, 61, 443–463, 2013.

FERRAZ, A. P. C. M. et al. **Taxonomia de Bloom: revisão teórica e apresentação das adequações do instrumento para definição de objetivos instrucionais.** Gest. Prod., São Carlos, 17(2), 2010, 421-431.

FILATRO, A. **Design instrucional contextualizado: educação e tecnologia**. São Paulo: SENAC, 2004.

GAGNE, R. M.; BRIGGS, L. J.; WAGER, W. W. **Principles of Instructional Design**. 4. ed. Florida: Harcourt Brace College Publishers, 1992.

HAIRBALL, 2017. Disponível em: <https://github.com/ucsb-cs-education/hairball>. Acesso em: setembro de 2017.

HAYDT, R. C. C. **Avaliação do processo ensino-aprendizagem**. São Paulo: Ed. Ática, 1988.

HONIG, W. L. **Teaching and assessing programming fundamentals for non majors with visual programming**. In: Proceedings of the 18th ACM conference on Innovation and technology in computer science education. Canterbury, UK, 2013. p. 40-45.

JAX-RS API, **Java™ API for RESTful Web Services (JAX-RS) delivers API for RESTful Web Services development in Java SE and Java EE.** 2017. Disponível em: < https://github.com/jax-rs>. Acesso em: julho de 2017.

JERSEY, **RESTful Web Services in Java.** 2017. Disponível em: < https://jersey.github.io/ >. Acesso em: julho de 2017.

KITCHENHAM, B. et al. **Systematic literature reviews in software engineering–a systematic literature review**. Information and software technology, 51(1), 2009.

LARMAN, C; BASILI, V. R. **Iterative and incremental developments. a brief history**. Computer, v. 36, n. 6, 2003.

LIBÂNEO, J. C. **Didática.** 29 ed. São Paulo: Cortez, 2009. 264 p.

LIBÂNEO, J. C. **Pedagogia e pedagogos, para quê?** São Paulo: Cortez, 1999.

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

MARCONDES, M. E. R. et al. **Materiais instrucionais numa perspectiva CTSA: uma análise de unidades didáticas produzidas por professores de química em formação continuada**. Investigações em Ensino de Ciências, v. 14, n. 2, 2009.

MEC, Secretaria de Educação Básica, Departamento de Políticas de Educação Infantil e Ensino Fundamental, Coordenação Geral do Ensino Fundamental. **Ensino Fundamental de Nove Anos – Orientações Gerais – Brasília, 2005.**

MEC; PCN. **Parâmetros Curriculares Nacionais: Introdução aos Parâmetros Curriculares Nacionais**, v. 1, Brasil, 1998.

MEC; PCN 5.1. **Parâmetros Curriculares Nacionais: História e Geografia (1ª a 4ª série)**, v. 5.1, Brasil, 1998.

MEC; PCN 5.2. **Parâmetros Curriculares Nacionais: História e Geografia (5ª a 8ª série),** v. 5.2, Brasil, 1998.

MEDEIROS, E. B. **Provas objetivas - técnicas de construção**. 3.ed. Rio de Janeiro, Fundação Getúlio Vargas, 1974.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. **Habits of programming in scratch**. In: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, Darmstadt, Alemanha, 2011.

MERCADO, L. P. L**. Novas tecnologias na educação: reflexões sobre a prática**. Brasil, 2002.

MORENO, R. **Decreasing cognitive load for novice students: Effects of explanatory versus corrective feedback in discovery-based multimedia. Instructional Science**, v. 32, n. 1, p. 99-113, 2004.

MORENO-LEÓN, J.; ROBLES, G.; **Computer programming as an educational tool in the English classroom a preliminary study**. In: Proceedings of 2015 IEEE Global Engineering Education Conference, Tallinn, Estonia, 2015.

MORENO-LEÓN, J.; ROBLES, G. **Analyze your Scratch projects with Dr. Scratch and assess your computational thinking skills**. In: Proceedings of the Scratch Conference, Amsterdam, Netherlands, 2015.

MORENO-LEÓN, J.; ROBLES, G.; ROMÁN-GONZÁLEZ, M. **Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking**. Revista de Educación a Distancia, v. 15, n. 46, 2015.

MYSQL, 2017. Disponível em: < https://www.mysql.com/>. Acesso em: agosto de 2017.

NEUNER, G. ET AL, **Pedagogía**. La Habana: libros para la educación,1981.

OLIVEIRA, A. Disponível em <http://www.cpt.com.br/pcn/pcn-parametros-curriculares-nacionais-do-6-ao-9-ano>. Acesso em: setembro de 2016.

OTA, G; MORIMOTO, Y; KATO, H. **Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking**

concepts. In: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, Chiba, Japão, 2016.

PETERSEN, K. et al. **Systematic Mapping Studies in Software Engineering**. 2008, p. 68-77.

PISKURICH, George M. **Rapid instructional design**: Learning ID fast and right. John Wiley & Sons, 2015.

DR. SCRATCH. **Dr.Scratch – Analise seus projetos Scratch aqui**. Disponível em: <http://www.drscratch.org/>. Acesso em: Setembro 2016.

RASHKOVITS, R; LAVY, I. **FACT: A Formative Assessment Criteria Tool for the Assessment of Students' Programming Tasks**. In: Proceedings of the World Congress on Engineering. Londres, UK, 2013.

RESNICK, M. et al. **Scratch: programming for all**. Communications of the ACM, v. 52, n. 11, 2009.

RESOLUÇÃO CME Nº02/2011. Disponível em: <http://www.pmf.sc.gov.br/arquivos/arquivos/pdf/07_05_2015_13.51.14.dfabda73d3789fb2cdb31482f7b5bb1c.pdf > Acesso em: junho de 2017.

RODRIGUEZ, Alex. **Restful web services: The basics**. IBM developerWorks, 2008.

ROUSE, M. **Static Analysis**. Disponível em: <http://searchwindevelopment.techtarget.com/definition/static-analysis?vgnextfmt=print>. Acesso em: outubro de 2016.

SBC, **Plano de Gestão para a SBC Biênio Agosto 2015 - Julho 2017**, 2015. Disponível em: <http://www.sbc.org.br/documentos-da-sbc/send/135-eleicoes/999-plano-de-gestao-para-a-sbc-bienio-agosto-2015-julho-2017> Acesso em: setembro de 2016.

SBC, **Referenciais de Formação em Computação: Educação Básica**, 2017. Disponível em: <http://www.sbc.org.br/files/ComputacaoEducacaoBasica-versaofinal-julho2017.pdf> Acesso em: setembro de 2017.

SCRATCH; MIT. **Scratch**, 2013. Disponível em: <http://scratch.mit.edu>. Acesso em: setembro 2016.

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

SIMPSON E.J. **The Classification of Educational Objectives in the Psychomotor Domain**. Washington, DC: Gryphon House, 1972.

SINTHSIRIMANA, S; PATCHARINPANJABUREE; **A Development of Computer Programming Analyzer: A Case Study on PHP Programming Language Institute for Innovative Learning**. Journal of Industrial and Intelligent Information, v. 1, n. 3, Bangkok, Thailand, 2013.

SNAP!; BERKELEY. **Snap!**, 2013. Disponível em: <http://snap.berkeley.edu>. Acesso em: novembro 2016.

STRIEWE, M.; GOEDICKE, M. **A review of static analysis approaches for programming exercises**. In: Proceedings of International Computer Assisted Assessment Conference, Duisburg, Alemanha, 2014.

The 2016 Top Programming Languages. Disponível em: http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages>. Acesso em: novembro de 2016.

TOWNHIDNEJAD, M.; HILBURN, T. B.; **Software Quality Across the Curriculum**. In: Proc. The 15th Conference on Software Engineering Education and Training, Covington, KY, USA, 2002.

TRUONG, N.; ROE, P.; BANCROFT, P. **Static analysis of students' Java programs.** In: Proceedings of the Sixth Australasian Conference on Computing Education, Australia, 2004.

VIEGA, J. et al. **ITS4: A static vulnerability scanner for C and C++ code.** In: Proceedings of the 16th Annual Conference On Computer Security Applications, Dulles, Virginia, USA, 2000.

WANGENHEIM, C. G.; NUNES, V. R.; SANTOS, G. D. Ensino de computação com scratch no ensino fundamental–um estudo de caso. **Revista Brasileira de Informática na Educação**, v. 22, n. 03, 2014.

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

WICHMANN, B. A. et al. Industrial perspective on static analysis. **Software Engineering Journal**, v. 10, n. 2, 1995.

WING, J. M. **Computational thinking**. Communications of the ACM, v. 49, n. 3, 2006.

YULIANTO, S. V.; LIEM, I. **Automatic grader for programming assignment using source code analyzer.** In: Proc. of International Conference on Data and Software Engineering, Bandung/Indonesia, 2014.

VALENTE, J. A.; DE ALMEIDA, F. J. Visão analítica da informática na educação no Brasil: a questão da formação do professor. **Brazilian Journal of Computers in Education**, v. 1, n. 1, 1997.

Zen, K., Iskandar, D.N.F.A., Linang, O. (2011). **Using Latent Semantic Analysis for automated grading programming assignments**. Proceedings of the 2011 International Conference on Semantic Technology and Information Retrieval, Putrajaya, Malaysia, pages 82 –88.

Anexo A – Convite, passo-a-passo e questionário para professores

Convite Professor

Olá,

Gostaríamos de convidar você para participar da avaliação do CodeMaster, uma ferramenta online que analisa e avalia projetos de apps programados no AppInventor e Snap! no contexto do ensino de programação em escolas.

A ferramenta foi desenvolvida no contexto do TCC de Matheus Faustino Demetrio sob a orientação da Profa. Christiane Gresse von Wangenheim realizado na iniciativa de Computação na Escola no INCoD/INE/UFSC.

Na avaliação estaremos solicitando a você que simule a avaliação de projetos. Disponibilizaremos a você um pacote de projetos AppInventor prontos para enviar ao sistema simulando a avaliação em massa de vários projetos de diferentes alunos, e ao final, o preenchimento de um questionário sobre a utilidade e usabilidade da ferramenta. No total, não deve levar mais do que 30 minutos.

Para facilitar a avaliação gostaríamos de realizar a avaliação junto com vocês para que possamos esclarecer quaisquer dúvidas que possam surgir no decorrer da tarefa. Caso você esteja de acordo, favor enviar e-mail para matheus.fdemetrio@gmail.com para agendar um horário para a avaliação a ser realizada na escola onde você atua.

A sua participação é voluntaria sem renumeração. Todos os seus dados serão tratados de forma sigilosa usados somente para fins de pesquisa.

Desde já, muito obrigado pela ajuda! O seu feedback é muito importante para nossa pesquisa.

Ao final estaremos disponibilizando a ferramenta gratuitamente no site da iniciativa Computação na Escola (http://www.computacaonaescola.ufsc.br) para qualquer interessado.

Att,

Matheus e Christiane

Cenário Professor

Imagine que você aplicou uma atividade ensinando a programação de software em uma turma na escola em que cada aluno deveria desenvolver um aplicativo simples para Smartphone Android utilizando o AppInventor. A atividade poderia ser feita em

duplas, trios ou sozinho. Assumimos, que ao todo 10 projetos de aplicativos AppInventor foram recebidos para avaliação. Para possibilitar a identificação dos projetos, Cada projeto já estava nomeado corretamente pelos próprios alunos, indicando os nomes dos alunos no nome do arquivo aia (NomeAluno1-NomeAluno2-NomeAluno3.aia). Agora você deseja avaliar estes 10 projetos de forma automatizada no CodeMaster para poder acompanhar o progresso de aprendizagem dos seus alunos e/ou dar uma nota.

Passo a Passo Professor

1. Baixar o pacote de projetos App Inventor no link: https://arquivos.ufsc.br/f/d9bad69a3c/?raw=1 e descompactar numa pasta de sua preferência. (Alternativamente se já usou App Inventor em uma das suas disciplinas você também pode usar os arquivos aia criados por seus alunos).
2. Abrir o CodeMaster no navegador de internet de sua preferência (Chrome, Firefox, Internet Explorer, Safari, etc): http://apps.computacaonaescola.ufsc.br:8080/
3. Clicar no menu superior na aba “Professor” e selecionar “Avaliar Projetos”. Você deve ser redirecionado para tela de login.
4. Se for o primeiro acesso clicar em “Ainda não sou cadastrado” logo abaixo do botão de login.
5. Criar cadastro com um e-mail válido. Você deve ser redirecionado para a tela de login se tudo ocorrer corretamente.
6. Fazer login com e-mail e senha cadastrados no passo anterior. Você será redirecionado para a tela de enviar projetos para análise que apenas professores tem acesso.
7. Digitar o nome da turma que preferir, selecionar o tipo de projetos que serão enviados (neste caso App Inventor) clicar em próximo.
8. Selecionar os conceitos que serão analisados dos projetos (neste caso aconselhamos deixar todos ativos). Clicar em próximo.
9. Selecionar todos os projetos descompactados no item 1. Clicar em Avaliar.
10. Deverá ser exibido na tela uma tabela com a pontuação obtida por cada projeto, bem como uma nota final de 0 a 10 e um nível.
11. Preencher o questionário do link: https://arquivos.ufsc.br/f/41551ce23c/?raw=1 e enviar para: matheus.fdemetrio@gmail.com

Questionário Professor

Nome * _____

Área de Atuação/Disciplinas * (pode ser mais do que uma resposta)

- Sala de Informática
- Língua Portuguesa
- Matemática
- Ciência (Biologia, Física, Química)
- Estudos sociais (História, Geografia)
- Inglês
- Educação Física
- Artes
- Outra: _____

Para qual nível do ensino você dá aulas?

- Ensino Fundamental I
- Ensino Fundamental II
- Ensino Médio

Você já ensinou computação/programação nas suas disciplinas? *

- Nunca
- 1 – 2 vezes
- 3-5 vezes
- Mais de 5 vezes

Qual ambiente de programação você usou no ensino de computação?
- App Inventor
- Scratch
- Snap!
- Outro: _____

Qual Sistema operacional você utilizou agora no teste do CodeMaster? *

- Linux
- Windows
- Mac
- Outro

Qual Navegador web você utilizou agora no teste do CodeMaster? *

- Mozilla Firefox
- Google Chrome

- Safari
- Microsoft Edge
- Outro

Quais projetos aia você usou no teste?

- Os 10 projetos disponibilizados por nós
- Projetos criados pelos meus alunos

Avaliação da ferramenta CodeMaster

Você acha a ferramenta CodeMaster útil no ensino de computação no ensino Básico? *

- Sim
- Não

Explique, porque:

_____

Você acha que existem aspectos/critérios de avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta? (Atualmente a ferramenta analisa: Telas, Interface de Usuário, Nomeação de Componentes, Eventos, Abstração de procedimentos, Laços, Condicionais, Listas, Persistência de dados, Sensores, Mídia, Social, Conectividade, Desenho e Animação)*

- Sim
- Não

Se sim, quais?

_____

Você acha que existem aspectos relevantes no (processo de) avaliação de projetos de programação no ensino de computação no ensino básico que não são suportados pela ferramenta? *

- Sim
- Não

Se sim, quais?

_____

Você acha que as informações disponibilizadas como resultado da avaliação são suficientes? *

- Sim
- Não

Se não, quais mais deveriam ser adicionadas?

_____

Você acha que na sua forma atual (fazer o upload de um conjunto de projetos de alunos identificando-os por seu nome no arquivo) é prático no seu dia-a-dia?

- Sim
- Não

Se não, o que deverá ser diferente?

_____

Você acha que a ferramenta possui elementos ambíguos ou difíceis de entender? *

- Sim
- Não

Se sim, quais?

A performance (tempo de resposta) do sistema é satisfatória? *

- Sim
- Não

Você observou algum erro (bug) em relação a funcionalidade da ferramenta?

- Sim
- Não

Se sim, qual(is)?

Você achou fácil de usar o sistema? *

- Sim
- Não

Se não, o que achou difícil?

Você observou algum erro (de ortografia/gramática) na interface da ferramenta? *

- Sim
- Não

Se sim, qual?

_____

Satisfação

Escala de resposta para cada um dos 10 itens: 1 (discordo totalmente), 2, 3, 4, 5 (concordo completamente)

Item	**Afirmação**	Discordo Totalmente **1**	**2**	**3**	**4**	Concordo completamente **5**
1	Eu penso que usarei este sistema com frequência.					
2	Acho o sistema desnecessariamente complexo.					
3	Penso que o sistema é fácil de usar.					
4	Acho que vou precisar da ajuda de um técnico para usar este sistema.					
5	Acho as funções deste sistema bem integradas.					
6	Encontro muitas inconsistências neste sistema.					
7	Imagino que as pessoas aprenderão rapidamente a usar este sistema.					
8	Não acho o sistema prático de usar.					
9	Senti-me confiante ao usar o sistema.					
10	Precisei aprender muitas coisas antes de ser capaz de operar o sistema.					

Comentários gerais

O que mais você gostou da ferramenta CodeMaster? *

_____

Alguma sugestão de melhoria referente a ferramenta CodeMaster?*

_____

Mais algum comentário?

Muito obrigado pela sua participação!

Anexo B – Convite, passo-a-passo e questionário para alunos

Convite Aluno

Olá,

Gostaríamos de convidar você apara participar de uma avaliação do CodeMaster, uma ferramenta online que analisa e avalia projetos de apps programados no AppInventor e Snap!.

A ferramenta foi desenvolvida no contexto do TCC de Matheus Faustino Demetrio sob a orientação da Profa. Christiane Gresse von Wangenheim realizado na iniciativa de Computação na Escola no INCoD/INE/UFSC.

Na avaliação estaremos solicitando a você simular a avaliação de um dos seus projetos programados no AppInventor, enviando o projeto e recebendo então a avaliação. No final, você deve responder um questionário sobre a utilidade e usabilidade da ferramenta. No total, não deve levar mais do que 30 minutos.

Em anexo segue um arquivo com todas as instruções (links de acesso e download do projeto a ser avaliado) bem como um passo a passo explicando como proceder a análise no CodeMaster. Sugerimos que você peça aos seus pais para ajudar em caso de qualquer dúvida.

O teste pode ser feito online na sua casa mesmo. Ao final, favor envie o questionário respondido via e-mail para: matheus.fdemetrio@gmail.com.

A sua participação é voluntaria sem remuneração. Todos os seus dados serão tratados de forma sigilosa usados somente para fins de pesquisa.

Desde já, muito obrigado pela ajuda! O seu feedback é muito importante para nossa pesquisa.

Ao final estaremos disponibilizando a ferramenta gratuitamente no site da iniciativa Computação na Escola (http://www.computacaonaescola.ufsc.br) para qualquer interessado.

Att,

Matheus e Christiane

Cenário Aluno

Imagine que você fez um app com App Inventor. Agora você gostaria de saber até que ponto você está dominando a programação de apps – se ainda está só iniciando isto ou já é um mestre em programação. Assim, você abre o CodeMaster e analisa o seu projeto.

Passo a passo Aluno

1. Baixar o projeto App Inventor no link: https://arquivos.ufsc.br/f/d500511586/?raw=1, guardar numa pasta de sua preferência. (Se preferir avaliar algum projeto criado por você no App Inventor, vá ao seu projeto no App Inventor, clique no menu superior em “Projetos” e em “Exportar o projeto selecionado (.aia) para o meu computador”.)
2. Abrir o CodeMaster no navegador de internet de sua preferência (Chrome, Firefox, Internet Explorer, Safari, etc.): http://apps.computacaonaescola.ufsc.br:8080/
3. Clicar no menu superior a opção “Aluno”.
4. Escolher o arquivo baixado no passo 1.
5. Clicar em Avaliar.
6. Resultado é exibido. Cada conceito analisado retorna uma pontuação de 0 a 3. Você recebe uma nota geral de 0 a 10 e um nível ninja.
7. Responder o questionário do link: https://arquivos.ufsc.br/f/581fde94e9/?raw=1 e enviar para o e-mail: matheus.fdemetrio@gmail.com.

Questionário Aluno

Nome *_____

Você está em que ano do ensino fundamental?*

- 1° ano
- 2° ano
- 3° ano
- 4° ano
- 5° ano
- 6° ano
- 7° ano
- 8° ano
- 9° ano

- Outro: _____

Você já programou um programa de software? *

- Nunca
- 1 – 2 vezes
- 3-5 vezes
- Mais de 5 vezes

Qual ambiente de programação você usou para programar? (Pode ser mais de uma resposta)
- App Inventor
- Scratch
- Snap!
- Outro: _____

Qual Sistema operacional você utilizou agora no teste do CodeMaster? *

- Linux
- Windows
- Mac
- Outro

Qual Navegador web você utilizou agora no teste do CodeMaster? *

- Mozilla Firefox
- Google Chrome
- Safari
- Microsoft Edge
- Outro

Avaliação da ferramenta CodeMaster

Você acha a ferramenta CodeMaster útil na aprendizagem de programação? *

- Sim
- Não

Explique, porque: _____

Você achou a nota justa? *

- Sim
- Não

Se não, Porque?

_____

Você acha que as informações disponibilizadas como resultado da avaliação são suficientes? *

- Sim
- Não

Se não, quais mais deveriam ser adicionadas?

_____

Você acha que a ferramenta possui elementos ambíguos ou difíceis de entender? *

- Sim
- Não

Se sim, quais?

A performance (tempo de resposta) do sistema é satisfatória? *

- Sim
- Não

Você observou algum erro (bug) em relação a funcionalidade da ferramenta?

- Sim
- Não

Se sim, qual(is)?

Você achou fácil de usar o sistema? *

- Sim
- Não

Se não, o que achou difícil?

Você observou algum erro (de ortografia/gramática) na interface da ferramenta? *

- Sim
- Não

Se sim, qual?

_____

Satisfação

Escala de resposta para cada um dos 10 itens: 1 (discordo totalmente), 2, 3, 4, 5 (concordo completamente)

Item	**Afirmação**	Discordo Totalmente **1**	**2**	**3**	**4**	Concordo completamente **5**
1	Eu penso que usarei este sistema com frequência.					
2	Acho o sistema desnecessariamente complexo.					
3	Penso que o sistema é fácil de usar.					
4	Acho que vou precisar da ajuda de um técnico para usar este sistema.					
5	Acho as funções deste sistema bem integradas.					
6	Encontro muitas inconsistências neste sistema.					
7	Imagino que as pessoas aprenderão rapidamente a usar este sistema.					
8	Não acho o sistema prático de usar.					
9	Senti-me confiante ao usar o sistema.					
10	Precisei aprender muitas coisas antes de ser capaz de operar o sistema.					

Comentários gerais

O que mais você gostou da ferramenta CodeMaster? *

_____

Alguma sugestão de melhoria referente a ferramenta CodeMaster?*

_____

Mais algum comentário?

Muito obrigado pela sua participação!

Anexo C – Artigo Sobre o Trabalho de Conclusão de Curso

CodeMaster – Análise e Avaliação Automática de Programas App Inventor

Matheus Faustino Demetrio
Departamento de Informática e Estatística, Universidade Federal de Santa Catarina
Florianópolis/SC, Brasil
matheus.demetrio@grad.ufsc.br

Resumo

O desenvolvimento do pensamento computacional é o tópico principal do currículo K-12 para educação. Muitas dessas experiências focam em ensinar programação usando linguagens de programação baseadas em blocos. Como parte dessas atividades, é importante para os estudantes receberem *feedback* sobre o aprendizado. Porém, na prática pode ser difícil prover um *feedback* individual, objetivo e consistente. Neste contexto, a análise e avaliação automática se torna importante. Enquanto existem diversos avaliadores para linguagens de programação baseadas em texto, o suporte para linguagens baseadas em blocos ainda é escasso. Este artigo apresenta o CodeMaster, uma ferramenta web gratuita que em um contexto de ensino baseado em problemas, permite analisar e avaliar automaticamente projetos programados com App Inventor. O CodeMaster utiliza uma rubrica para mensurar o pensamento computacional baseado numa análise estática do código. Os estudantes podem utilizar a ferramenta para obter um feedback e encoraja-los a melhorar as suas competências em programação. Também pode ser utilizado por professores para avaliar turma inteiras, facilitando o fluxo de trabalho.

Palavras-chave: Pensamento computacional, Programação, Análise, Avaliação, App Inventor

1. Introdução

Pensamento Computacional é a competência que envolve resolver problemas, projetar sistemas, e entender o comportamento humano, formados por conceitos fundamentais da ciência da computação (Wing, 2006). É considerado a competência chave para a atual geração de estudantes em um mundo extremamente influenciado pelos princípios da

computação (Wing, 2006). Por esse motivo, ensinar o pensamento computacional tem sido o foco por todo o mundo (Grover and Pea, 2013) (Kafai and Burke, 2013) (Resnick *et al.*, 2009). Muitas dessas iniciativas focam em ensinar programação, que não é apenas uma parte fundamental da computação, mas também uma importante ferramenta para suporte a tarefas cognitivas que envolvem o pensamento computacional (Grover and Pea, 2013). Linguagens de programação baseada em blocos encorajam e motivam a aprendizagem de conceitos de programação, reduzem a carga cognitiva e permitem o foco na lógica e estruturas envolvidas na programação ao invés do aprendizado de complexas sintaxes de linguagens de programação baseadas em texto (Kelleher and Pausch, 2005)(Maiorana *et al.*, 2015)(Grover *et al.*, 2015). Muitas atividades do pensamento computacional focam em criar soluções para problemas do mundo real, em que as soluções são artefatos de software, como games/animações em tópicos interdisciplinares ou aplicativos móveis para resolver problemas na comunidade (Monroy-Hernández and Resnick, 2008) (Fee and Holland-Minkley, 2010). Neste contexto, as atividades não possuem uma única solução correta.

Um elemento crucial no processo de aprendizagem é a avaliação e feedback (Hattie and Timperley, 2007) (Shute, 2008) (Black and Wiliam, 1998). A avaliação guia a aprendizagem e prove um feedback tanto para o estudante quanto para o professor. Para uma aprendizagem efetiva, os estudantes precisam saber o seu nível de performance em uma tarefa, como relacionar a sua performance a uma boa performance e o que fazer para diminuir a diferença entre elas. (Sadler, 1989)

Não existe consenso nas estratégias para avaliar conceitos do pensamento computacional (Brennan and Resnick, 2012) (Grover *et al.*, 2014). A avaliação do pensamento computacional é particularmente complexa pela natureza abstrata da construção mensurada. Muitos autores têm proposto diferentes aproximações e frameworks para tentar guiar a avaliação dessa competência de diferentes formas, incluindo a avaliação de artefatos de softwares criados por estudantes (Brennan and Resnick, 2012). As avaliações de softwares podem abranger diversos aspectos de qualidade, como corretude, complexidade, reuso, conformidade com padrões de programação, etc.

Outro problema que complica a avaliação do pensamento computacional na pratica educacional é que a avaliação manual de trabalhos de programação requer recursos substanciais em se tratando de tempo e pessoas. Isso pode causar uma baixa escalabilidade no ensino de computação a um grande número de estudantes (Eseryel *et al.*, 2013) (Romli *et al.*, 2010) (Ala-Mutka, 2005). Além disso, muitos professores de outras disciplinas introduzem o ensino de computação de forma interdisciplinar em suas turmas, com isso, esses professores, são desafiados em respeito a avaliação. Desse modo, a avaliação manual pode permitir que erros aconteçam na avaliação dos estudantes, bem como inconsistência, fadiga e favoritismo (Zen *et al.*, 2011).

Neste contexto, a adoção de ferramentas de avaliação automática pode ser benéfica, facilitando o fluxo de trabalho do professor e permitindo mais tempo para outras atividades com os estudantes (Ala-Mutka and Järvinen, 2004). Isso também pode ajudar a manter a consistência e acurácia dos resultados das avaliações, bem como eliminar distorções. Para os estudantes, esse modo pode prover um feedback imediato para seus programas, permitindo que eles façam progresso sem um professor ao seu lado (Douce *et al.*, 2005) (Wilcox, 2016) (Yaday *et al.*, 2015). Então, a avaliação automática pode ser benéfica para ambos, estudantes e professores, melhorando a educação em computação.

Desta forma, apresentamos o CodeMaster, uma ferramenta web gratuita que analisa projetos App Inventor e oferece um feedback para professores e estudantes atribuindo uma nota do pensamento computacional aos projetos. Estudantes podem usar este feedback para melhorar seus programas e suas competências em programação. O objetivo do CodeMaster é prover suporte na analise e avaliação de critérios que podem ser automatizados de modo que os professores ficam livres para completar a avaliação manualmente em critérios importantes como criatividade e inovação.

2. Fundamentação

Este capítulo apresenta os conceitos referentes a teoria da aprendizagem e do ensino, bem como o princípio de uma unidade instrucional e *design* instrucional. Também são apresentados conceitos de análise de código e uma visão geral sobre o ambiente de programação App Inventor.

2.1 Aprendizagem e Ensino

A aprendizagem é o processo de assimilação de conhecimentos, habilidades ou valores por meio do estudo, da prática, da experiência, do raciocínio, ou da observação. A aprendizagem está presente em qualquer atividade humana em que possamos aprender algo (LIBANEO, 1999). Aprendizagem pode acontecer de diversas formas, nas relações interpessoais do dia-a-dia ou sozinho, mas para aumentar as chances de a aprendizagem ser bem-sucedida, ela é institucionalizada por meio do ensino.

O ensino é o processo de transmissão e apropriação de uma competência (NEUNERET et al., 1981). A competência pode ser vista como sendo a junção de conhecimentos, de habilidades e de atitudes. No ensino, usualmente, existem como atores, o instrutor e o aprendiz. Assim, os objetivos da interação entre os atores, resulta no ato de aprender do aprendiz.

No ensino cria-se e usa-se unidades instrucionais (UIs) guiando e apoiando os instrutores (MARCONDES et al., 2009). Uma unidade instrucional pode ser uma aula, um curso, uma oficina, exercícios, atividades, organizado em uma sequência lógica, que guie de forma precisa e facilitadora o instrutor. A unidade instrucional apresenta uma visão concisa ao instrutor do objetivo da aprendizagem, do conteúdo e da sequência e os métodos de ensino a ser ensinado, o que inclui os materiais e ferramentas que serão utilizados (VON WANGENHEIM et. al., 2014).

Porém, para assegurar que a aprendizagem seja efetiva e eficiente e ao mesmo tempo motivadora e divertida, é importante que as unidades instrucionais sejam desenvolvidas e avaliadas seguindo um processo e sistemática de design instrucional (VON WANGENHEIM et. al., 2014).

No ensino da computação, é possível inferir as competências adquiridas durante o processo de aprendizagem por meio da análise/avaliação de trabalhos práticos de programação. Nos trabalhos é possível identificar se os alunos têm capacidade de colocar em prática os conceitos teóricos, mostrando habilidade em organizar, sintetizar e aplicar a informação e a competência adquirida, e a habilidade em gerenciar os recursos disponíveis (MORENO-LEÓN et al., 2015).

Porém, a análise destes trabalhos práticos na avaliação, tipicamente, requer um esforço considerável e competência do professor na área de conhecimento (MEDEIROS, 1974).

2.2 Análise de Código Aplicada à Educação

O objetivo da análise de código aplicada a educação é analisar e avaliar o código de um trabalho prático desenvolvido pelo aluno no processo de ensino de computação.

Não há uma definição única sobre quais aspectos devem ser analisados pelo analisador e de modo geral pelo analisador estático. Porém, como em geral a avaliação de trabalhos de alunos foca na avaliação do grau de atendimento dos objetivos de aprendizagem. Assim, estes aspectos a serem avaliados devem ser derivados dos objetivos.

Existem também estudos que perceberam alguns hábitos de programação em estudantes que são contrários aos hábitos aceitáveis na computação (MORENO & ROBLES, 2015). Voltado ao foco do presente artigo os aspectos importantes que, tipicamente, devem ser analisados neste contexto são (RASHKOVITS & LAVY, 2013) (MORENO & ROBLES, 2015):

- Abstração e Modularização: O código deve ser eficientemente organizado em classes e as classes devem ser organizadas em hierarquias. Cada classe representa um único conceito e tem todos os atributos e métodos necessários.
- Projeto de métodos: Um método deve ser relativamente pequeno e deve executar uma única tarefa ou um conjunto altamente relacionado de pequenas tarefas.
- Legibilidade de código: O nome de classes, variáveis e métodos deve ser significativo para facilitar o entendimento do código.
- Paralelismo: O código deve priorizar o paralelismo na execução de atividades não relacionadas ou que não dependem uma da outra para execução.
- Sincronização: O código deve conter componentes capazes de sincronizar a execução de vários elementos dinâmicos da aplicação.
- Controle de fluxo: As instruções, expressões e chamadas de métodos devem ser corretamente ordenadas. O uso de condicionais, laços e chamada de sub-rotinas deve ser eficiente.
- Interatividade com usuário: A aplicação deve ser interativa com o usuário. O usuário deve ser participante.
- Representação de dados: A aplicação deve representar e armazenar os dados coletados de forma eficiente, e de forma a garantir a consistência dos dados.

Além desses aspectos, em aplicativos para dispositivos moveis, pode-se incluir a análise de alguns pontos específicos que são, geralmente, encontrados em apenas aplicações deste gênero. Estes aspectos incluem (SHERMAN, 2015):

- Compartilhamento de dados;
- Uso de serviços web públicos;
- Uso de acelerômetro e sensores de orientação;
- Reconhecimento de localização.

No foco do presente artigo necessita-se então da medição destes aspectos com base em trabalhos práticos, que possuam várias possíveis soluções a serem desenvolvidas pelos alunos. Neste caso a análise e avaliação de código requer mais flexibilidade, pois, possivelmente existem diversas soluções corretas ao problema. Assim, se faz necessário um analisador de código que possa fazer uma análise mais flexível e livre do código.

Neste contexto, a análise e avaliação do programa baseia-se no pressuposto de que certos atributos mensuráveis podem ser extraídos do programa, avaliando se os alunos aprenderam o que era esperado utilizando uma rubrica. A rubrica usa medidas descritivas para separar os níveis de desempenho em uma determinada tarefa, delineando os vários

critérios associados às atividades de aprendizado (WHITTAKER et al., 2001). Por meio da rubrica a pontuação de cada critério é definida, permitindo um feedback instrucional.

O feedback instrucional, resultado da avaliação, tipicamente inclui uma nota e um feedback ao aluno. No Brasil não existe um consenso e nem regra de como as notas devem ser dadas às provas e trabalhos práticos nas escolas. Cada estado ou município fica encarregado de definir qual padrão seguir, e mesmo assim podem haver diferenças entre a rede estadual e municipal além de pública e privada. Como exemplo na cidade de Florianópolis - SC a RESOLUÇÃO CME Nº02/2011 Art. 7º diz que as notas podem ser em "[...]parecer descritivo que revele o diagnóstico do processo de aprendizagem" e em número de 1 a 10.

O feedback é importante no contexto educacional para facilitar aprendizagem (MORENO, 2004). O feedback é uma técnica para orientar os alunos a pensarem sobre suas respostas e fornecer informações para direcionar a mudança de sua forma de pensar e agir (BILAL et al., 2012). Pode ser considerado uma resposta as ações do aluno, indicando a exatidão das respostas ou sendo mais amplo provendo dicas, sugestões e exemplos relacionados ao conteúdo (BLACK & WILIAN, 1998). Portanto é importante que ao final da avaliação sejam dados uma nota e um feedback ao aluno.

2.3 App Inventor

Existem muitas linguagens de programação hoje em dia, cada uma com suas peculiaridades e objetivos (IEEE, 2016). A maior parte das linguagens de programação hoje utilizadas no desenvolvimento profissional são linguagens de programação textual. Porém, com o objetivo de engajar crianças e jovens na área da computação e para facilitar a aprendizagem de computação nesta faixa etária, são indicados o uso de linguagens de programação visuais (RESNICK et al., 2009).

Linguagens de programação visuais baseado em blocos permitem criar sistemas de software ou aplicativos arrastando e encaixando blocos. Nas linguagens visuais todos os componentes das linguagens já estão definidos, normalmente, em forma de figuras ou blocos, o usuário deverá colocar esses blocos numa ordem que faça sentido ao programa e então a aplicação começa a ser programada. Desta maneira ele não precisará escrever linhas de código textuais, eliminando a necessidade de se entender e memorizar as complexas sintaxes de linguagens textuais.

Umas das linguagens de programação visual é o Blockly (BLOCKLY, 2017). Criado pela Google, tem como objetivo ser uma biblioteca que sirva como base para o desenvolvimento de ambientes de programação.

O App Inventor (APP INVENTOR, 2017) é um ambiente de programação visual criado pela Google e atualmente mantido pelo Instituto de Tecnologia de Massachusetts (MIT). O App Inventor utiliza a linguagem de programação visual Blockly para permitir a criação de aplicativos para smartphones e tablets com sistema operacional Android. O App Inventor, por meio de uma interface gráfica simples, permite que pessoas inexperientes em programação ou até mesmo crianças tenham a habilidade de criar do mais básico ao mais completo aplicativo em uma hora ou menos (APP INVENTOR, 2017). O objetivo da criação do App Inventor foi democratizar o desenvolvimento de software dando apoio a todas as pessoas, principalmente aos jovens, na transição de simples consumidores de tecnologia para criadores de tecnologia (APP INVENTOR, 2017). Em 2016 o App Inventor já era utilizado por mais de 5 milhões de usuários em mais de 195 países (APP INVENTOR, 2017).

O App Inventor permite que o projeto de app criado seja exportado em um arquivo compactado no formato aia. Na versão nb162a do App Inventor, este arquivo inclui todas as informações do app, como imagens e sons utilizados e os códigos produzidos pelos blocos da lógica e componentes visuais presentes no app. Os códigos referentes a parte visual do app ficam registrados em arquivos no formato scm que por sua vez encapsulam uma estrutura json que contém todos os componentes utilizados. Os códigos produzidos pela parte lógica do app ficam registrados em arquivos no formato bky que por sua vez encapsulam estruturas xml com toda a lógica implementada.

3. CodeMaster

Por meio da análise do estado da arte é possível identificar a necessidade de se desenvolver mais ferramentas de análise e avaliação de código para linguagens de programação visual. Portanto, a proposta é o desenvolvimento de uma ferramenta de análise e avaliação de código voltada especificamente para o App Inventor com foco no ensino de programação para o Ensino Básico. O objetivo é desenvolver uma ferramenta de análise de código do App Inventor no contexto de ensino, o CodeMaster - App Inventor. A ferramenta deve permitir a análise e avaliação automática dos trabalhos práticos desenvolvidos pelos alunos, fornecendo um *feedback* instrucional.

3.1 Modelo Conceitual

Com base nas ferramentas de análise e avaliação de código para linguagens de programação visual existentes e nas experiências práticas da iniciativa Computação na Escola são identificadas as principais funcionalidades da ferramenta:

- Permitir o upload de projetos do App Inventor versão *nb162a* no formato .aia;

- Analisar o projeto que contém o código da aplicação nos formatos *bky* e *scm*;
- Avaliar um projeto individual retornando um *feedback* instrucional incluindo nota, pontuação total, pontuação por critério e nível ninja;
- Apresentar as avaliações dos projetos ao professor em uma tabela com todos os projetos analisados e podendo exportar esses dados em planilha;
- Apresentar a avaliação do projeto ao aluno.
- Apresentar dados gerais e anônimos sobre as avaliações já realizadas pelo CodeMaster a pesquisadores interessados.

O CodeMaster deve avaliar, com base no Dr. Scratch (DR. SCRATCH, 2017), na rubrica para avaliar *apps* proposta por Sherman (2015) e no *framework* do pensamento computacional (BRENNAN & RESNICK, 2012), os conceitos do código conforme apresentado na Tabela 18.

Tabela 34 - Conceitos do Programação Analisados pelo CodeMaster e Rubrica de Avaliação

Critério	**Nível de Performance**			
	0	**1**	**2**	**3**
Telas	Apenas uma tela com componentes visuais e que seu estado não se altera com a execução do app (tela informativa).	Apenas uma tela com componentes visuais que se alteram com a execução do app.	Pelo menos duas telas e uma delas altera seu estado com a execução do app.	2 ou mais telas e pelo menos 2 delas alteram seus estados com a execução do app.
Interface de Usuário	Utiliza apenas 1 elemento visual sem alinhamento.	Utiliza 2 ou mais elementos visuais sem alinhamento.	Utiliza mais de 5 elementos visuais com 1 tipo de alinhamento.	Utiliza mais de 5 elementos visuais com 2 ou mais elementos de alinhamento.
Nomeação: Variáveis e procedimentos	Nenhum ou poucos nomes são alterado do padrão. (menos do que 10%)	De 10 a 25% dos nomes são alterados do padrão.	De 26 a 75% dos nomes são alterados do padrão.	Mais de 76% dos nomes são alterados do padrão.
Eventos	Nenhum manipulador de evento é usado (ex. On click).	1 tipo de manipuladores de eventos é usado.	2 tipos de manipuladores de eventos são usados.	Mais de 2 tipos de manipuladores de eventos são usados.
Abstração de procedimentos	Não existem procedimentos.	Existe exatamente um procedimento e sua chamada.	Existe mais de um procedimento.	Existem procedimentos tanto para organização quanto para reuso. (Mais chamadas de procedimentos do que procedimentos).
Laços	Não usa laços	Usa “While” (laço simples)	Usa “For each” (variável simples)	Usa “For each” (item de lista)
Condicionais	Não usa condicionais.	Usa apenas “if’s” simples.	Usa apenas “if then else”.	Usa um ou mais “if - else if”.
Operações Lógicas e Matemáticas	Não usa operadores	Usa um tipo de operador.	Usa dois tipos de operadores.	Usa mais de 2 tipos de operadores.

Listas	Não usa listas.	Usa uma lista unidimensional.	Usa mais de uma lista unidimensional.	Usa uma lista de tuplas (map).
Persistência de dados	Dados são armazenados em variáveis ou propriedades de componentes e não tem persistência quando app é fechado.	Usa persistência em arquivo (File ou Fusion Tables).	Usa algum dos bancos de dados locais do App Inventor (TinyDB).	Usa uma base de dados web tinywebdb ou Firebase do App Inventor (firebase ou TinyWebDB).
Sensores	Sem uso de sensores.	Usa um tipo de sensor.	Usa 2 tipos de sensores.	Usa mais de 2 tipos de sensores.
Mídia	Sem uso de mídias.	Usa um tipo de Mídia.	Usa 2 tipos de Mídia.	Usa mais de 2 tipos de mídia.
Social	Sem uso de componentes sociais.	Usa um tipo de componente social.	Usa 2 componentes visuais.	Usa mais de 2 componentes visuais.
Conectividade	Sem uso de componentes de conectividade.	Uso de iniciador de atividades (chama início de outro app no celular).	Uso de conexão bluetooth.	Uso de conexão web, baixo nível.
Desenho e Animação	Sem uso de desenho e animação.	Uso de área sensível ao toque.	Uso de animação com bolinha pré-definida.	Uso de animação com imagem.

Como resultado da análise dos critérios o sistema apresenta uma nota e um nível de competência ao aluno.

Nota. Com base nos conceitos apresentados na Tabela 1, o CodeMaster apresenta uma pontuação para cada critério. A nota final do projeto é definida numa escala de 0 a 10, alinhando a escala de notas tipicamente utilizados no ensino básico no Brasil. Para se obter a nota final, é definida a seguinte fórmula:

$$Nota\ final = \frac{Pontuação\ total\ recebida}{Pontuação\ máxima\ possível} * 10$$

Nível de competência. Para tornar essa nota mais divertida e agradável aos jovens, o resultado é transformado na apresentação de nível de competência representado por faixas de ninjas em cores diferentes. Cada faixa de valores de possíveis notas é mapeado a cores de faixa de ninja, como mostrado na tabela 2.

Tabela 35 - Mapeamento Nota e Faixa Ninja

Nota	Faixa Ninja
0 - 0.9	Branca
1.0 – 1.9	Amarela
2.0 – 2.9	Laranja

3.0 – 3.9	Vermelha
4.0 – 4.9	Roxa
5.0 – 5.9	Azul
6.0 – 6.9	Turquesa
7.0 – 7.9	Verde
8.0 – 8.9	Marrom
9.0 - 10.0	Preta

Feedback. O *feedback* é um conjunto de informações retornadas pela ferramenta após a análise e avaliação de código. Tem o intuito de guiar o aluno no processo de aprendizagem e o professor no processo de ensino. No CodeMaster o *feedback* é apresentado de duas formas, dependendo do modo de uso.

Modo de uso. O CodeMaster tem dois modos de uso. No modo individual o aluno submete seu próprio código à análise sem necessitar de cadastro. No modo individual, o *feedback* inclui:

- A pontuação total e a nota (em termos numéricos e por meio da faixa do ninja usando a imagem do ninja);
- A pontuação por critério de avaliação;
- A rubrica de avaliação.

As pontuações e nota dão ao aluno o *feedback* em termos da performance atingida no desenvolvimento do projeto analisado. A rubrica deve ser utilizada para guiar a melhoria da performance do aluno indicando como poderá melhorar a sua pontuação nos diversos critérios analisados.

No modo professor o mesmo pode submeter um conjunto de projetos App Inventor à ferramenta. Este conjunto de projetos representa uma turma. O feedback é dado em relação a turma toda, indicando por aluno:

- Pontuação total e a nota (em termos numéricos e por meio da cor da faixa do ninja);
- Pontuação por critério de avaliação;
- Médias da turma por critério de avaliação, pontuações totais e notas;
- A rubrica de avaliação.

Por meio dos resultados da avaliação apresentados, o professor analisa o nível de competência que o aluno se encontra, deste modo estará fazendo a avaliação somativa do aluno. Além disso, analisando os resultados globais da turma o professor consegue fazer a avaliação formativa, que diz respeito a avaliação da qualidade da unidade instrucional aplicada ao ensino.

3.2 Implementação

Com base no modelo conceitual, o CodeMaster foi desenvolvido como uma aplicação web. A ferramenta automatiza a análise e avaliação de projetos de programação do App Inventor. A Figura 1 apresenta uma visão geral dos casos de uso implementados pelo CodeMaster.

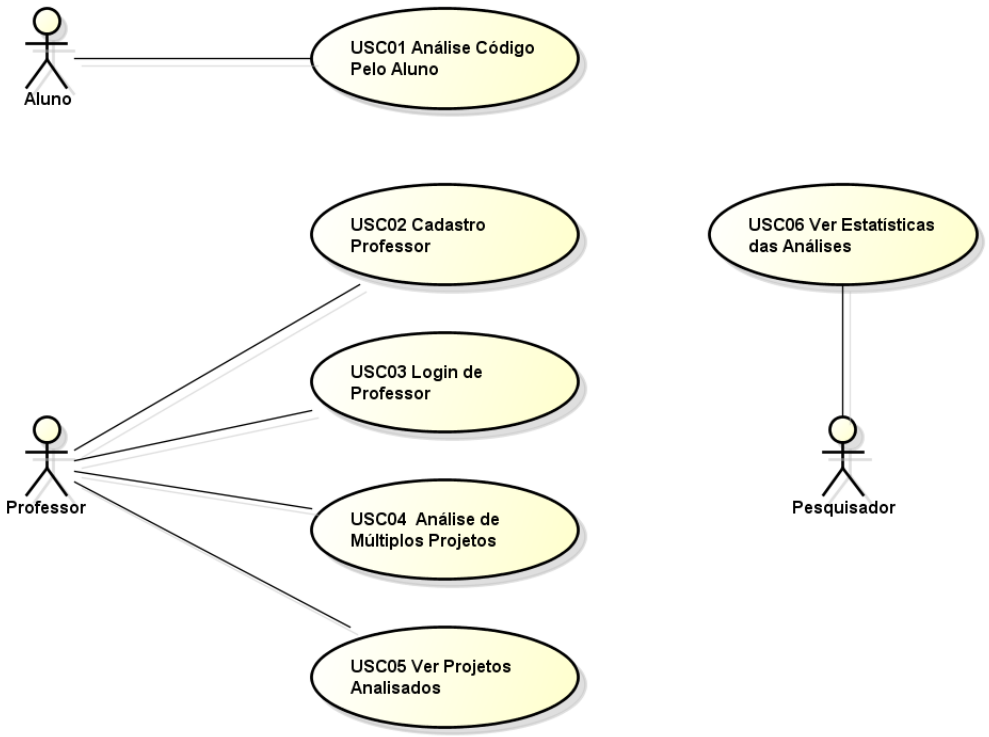


Figura 26 - Diagrama de Casos de Uso do CodeMaster

O modelo arquitetural do CodeMaster foi definido com o objetivo de separar a camada de apresentação da camada de avaliação em diferentes módulos para fazer a aplicação escalável em longo prazo e permitir a conexão direta ao avaliador por outras aplicações no futuro. O módulo de avaliação é um serviço web REST responsável por receber o projeto, suas configurações e retornar o resultado da avaliação. Ele foi implementado utilizando o framework Jersey que usa a API JAX-RS para abstrair os detalhes de baixo nível da implementação da comunicação entre os servidores e simplificar a implementação do serviço REST. O módulo de apresentação é responsável pela interface de usuário, registro de professores e turmas, submissão de projetos e apresentação dos resultados. A Figura 2 apresenta uma visão geral da arquitetura dos componentes de sistema do CodeMaster.

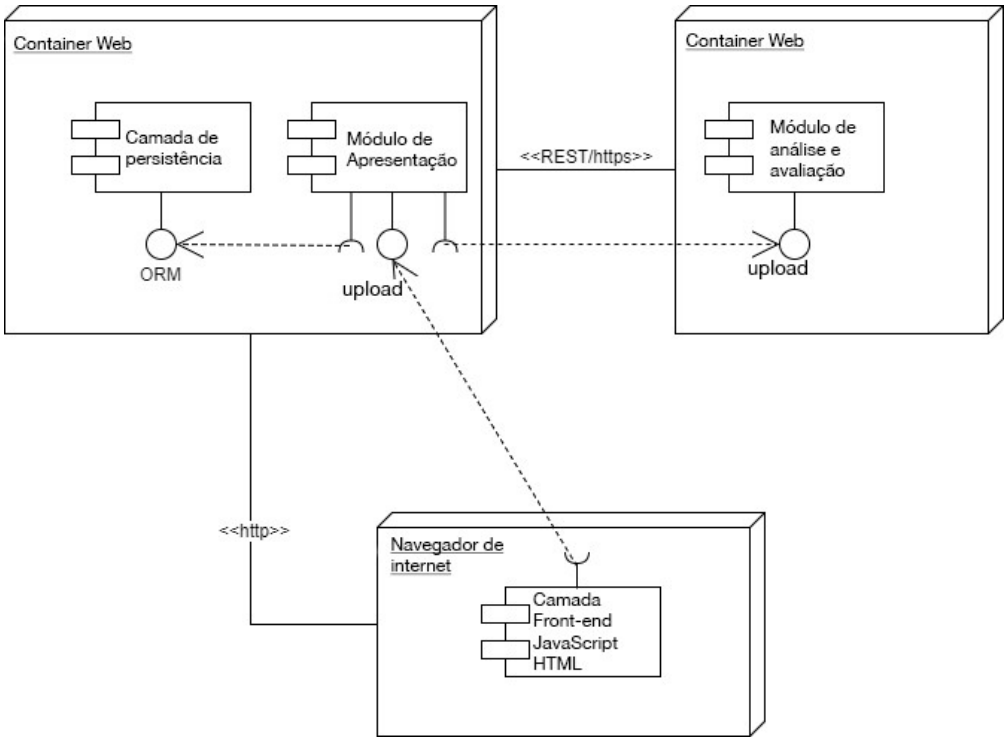


Figura 27 - Diagrama de Componentes do CodeMaster

Todo o *backend* foi implementado na linguagem de programação Java 8, rodando em um servidor de aplicação Apache Tomcat 8 sobre o sistema operacional Ubuntu 16, que foi a infraestrutura disponível. O *frontend* foi implementado em JavaScript usando a biblioteca Bootstrap com layout adicional. O banco de dados utilizado foi o MySQL 5.7, também disponível em nossa infraestrutura e é capaz de suportar a demanda inicial estimada. O CodeMaster está disponível online em português e inglês em: http://apps.computacaonaescola.ufsc.br:8080.

4. Conclusão

O objetivo principal deste artigo foi descrever o desenvolvimento de uma ferramenta web para análise e avaliação de código gerado pelo App Inventor para dar suporte ao ensino de computação ao Ensino Básico. Neste contexto, foi feita a análise da fundamentação teórica. Também foi realizada a análise do estado da arte identificando a falta de ferramentas de análise de código para suporte ao ensino de computação de modo generalizado. Deste modo, baseando-se na fundamentação teórica e na análise do estado da arte, foi elaborado um modelo conceitual incluindo a definição de uma rubrica para análise de código do App Inventor e foi desenvolvida uma ferramenta que analisa e avalia automaticamente projetos de apps criados com o App Inventor. Os resultados preliminares da avaliação do CodeMaster -

App Inventor fornecem uma indicação inicial da sua corretude, utilidade, funcionalidade, desempenho e usabilidade muito positiva.

O CodeMaster permite o cadastro de professores e a análise e avaliação de múltiplos projetos em lote, podendo ser agrupados por turmas de alunos. Além do serviço disponibilizado a professores que querem ensinar programação para seus alunos, a ferramenta permite que alunos possam avaliar seus projetos e receberem um feedback de como está o seu nível de competência em programação por meio de notas e faixas de ninja que tornam o sistema mais amigável para os jovens aprendizes.

Com o CodeMaster – App Inventor, existe hoje uma ferramenta robusta e única para análise e avaliação automatizada de códigos produzidos no ambiente de programação visual App Inventor, dando suporte a professores e alunos no ensino e aprendizagem de computação no Ensino Básico.

Referências

Ala-Mutka, K. M. (2005). A Survey of Automated Assessment Approaches for Programming Assignments. Computer Science Education, 15(2), 83–102.

Ala-Mutka, K. M., Järvinen, H.-M. (2004). Assessment process for programming assignments. Proceedings of IEEE Int. Conference on Advanced Learning Technologies, Joensuu, Finland, 181–185.

APP INVENTOR, 2017. Disponível em: <http://appinventor.mit.edu/explore/>. Acesso em: junho de 2017.

BILAL, M., CHAN, P., MEDDINGS, F., & KONSTADOPOULOU, A. SCORE: An advanced assessment and feedback framework with a universal marking scheme in higher education. In Proc. of the Int. Conf. on Education and e-Learning Innovations, Sousse/Tunísia, 2012, 1-6.

Black, P. Wiliam, D. (1998). Assessment and classroom learning. Assessment in Education: Principles, Policy & Practice, 5(1), 7–74.

BLOCKLY, 2017. Disponível em: <https://developers.google.com/blockly/>. Acesso em: setembro de 2017.

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

Douce, C., Livingstone, D., Orwell, J. (2005). Automatic test-based assessment of programming: A review. Journal on Educational Resourses in Computing, 5(3), no.4.

Eseryel, D., Ifenthaler, D., Xun, G. (2013). Validation study of a method for assessing complex ill-structured problem solving by using causal representations. Educational Technology and Research, 61, 443–463.

Fee, S. B., Holland-Minkley, A. M. (2010). Teaching Computer Science through Problems, not Solutions. Computer Science Education, 2, 129-144.

Grover, S., Cooper, S., Pea, R. (2014). Assessing Computational Learning in K-12. Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education, Uppsala, Sweden, 57-62.

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

Grover, S., Pea, R., Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. Journal Computer Science Education, 25(2), 199-237.

Hattie, J., Timperley, H. (2007). The power of feedback. Review of educational research, 77(1), 81-112.

Kafai, Y., Burke, Q. (2013). Computer programming goes back to school. Phi Deltan Kappan, 95(1), 61–65.

Kelleher, C., Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, 37(2), 83–137.

LIBÂNEO, J. C. Pedagogia e pedagogos, para quê? São Paulo: Cortez, 1999.

Maiorana, F., Giordano, D., & Morelli, R., (2015). Quizly: A live coding assessment platform for App Inventor. Proceedings of IEEE Blocks and Beyond Workshop,Atlanta, GA, USA, 25-30.

MARCONDES, M. E. R. et al. Materiais instrucionais numa perspectiva CTSA: uma análise de unidades didáticas produzidas por professores de química em formação continuada. Investigações em Ensino de Ciências, v. 14, n. 2, 2009.

MEDEIROS, E. B. Provas objetivas - técnicas de construção. 3.ed. Rio de Janeiro, Fundação Getúlio Vargas, 1974.

Monroy-Hernández, A.; Resnick, M. (2008). Empowering kids to create and share programmable media Interactions. 15(2), 50-53.
MORENO, R. Decreasing cognitive load for novice students: Effects of explanatory versus corrective feedback in discovery-based multimedia. Instructional Science, v. 32, n. 1, p. 99-113, 2004.
MORENO-LEÓN, J.; ROBLES, G.; Computer programming as an educational tool in the English classroom a preliminary study. In: Proceedings of 2015 IEEE Global Engineering Education Conference, Tallinn, Estonia, 2015.
NEUNER, G. ET AL, Pedagogía. La Habana: libros para la educación,1981.
RASHKOVITS, R; LAVY, I. FACT: A Formative Assessment Criteria Tool for the Assessment of Students' Programming Tasks. In: Proceedings of the World Congress on Engineering. Londres, UK, 2013.
Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K. (2009). Scratch: Programming for all. Communications of the ACM, 52(11), 60–67.
Romli, R., Sulaiman, S., Zamli, K. Z. (2010). Automatic programming assessment and test data generation - a review on its approaches. Proceedings of International Symposium in Information Technology, Kuala Lumpur, Malaysia, 1186–1192.
Sadler, D. R. (1989). Formative assessment and the design of instructional systems, Instructional Science, 18(2), 119-144.
Shute. V. J. (2008). Focus on formative feedback. Review of Educational Research, 78(1),153-189.
WANGENHEIM, C. G.; NUNES, V. R.; SANTOS, G. D. Ensino de computação com scratch no ensino fundamental–um estudo de caso. Revista Brasileira de Informática na Educação, v. 22, n. 03, 2014.
WHITTAKER, C. R., SALEND, S. J., DUHANEY, D. Creating instructional rubrics for inclusive classrooms. Teaching Exceptional Children, 34(2), 8-13, 2001.
Wilcox, C. (2016). Testing Strategies for the Automated Grading of Student Programs. Proceedings of the 47th ACM Technical Symposium on Computing Science Education, Memphis, Tennessee, USA, 437-442.
Wing, J. (2006). Computational Thinking. Communications of the ACM, 49(3), 33-36.
Yadav, A., Burkhart, D., Moix, D., Snow, E., Bandaru, P., Clayborn, L. (2015). Sowing the Seeds: A Landscape Study on Assessment in Secondary Computer Science Education. Proceedings of CSTA Annual Conference, Grapevine, TX, USA.
Zen, K., Iskandar, D.N.F.A., Linang, O. (2011). Using Latent Semantic Analysis for automated grading programming assignments. Proceedings of the 2011 International Conference on Semantic Technology and Information Retrieval, Kuala Lumpur, Malaysia, 82 –88.