**CENTRO PAULA SOUZA**

**FACULDADE DE TECNOLOGIA DE MOGI MIRIM**

Derik Teruo Barbosa Hito

Giovani Kelvin da Rocha

Lucas Oliveira de Jesus

Learning Vision: Aplicação de Machine Learning para auxilio a visão

**MOGI MIRIM - SP**

**2018**

**Derik Teruo Barbosa Hito**

**Giovani Kelvin da Rocha**

**Lucas Oliveira de Jesus**

**Learning Vision: Aplicação de Machine Learning para auxilio a visão**

Trabalho de Graduação apresentado ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Mogi Mirim como pré-requisito para a obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador:** Prof. Me. Marcio Rodrigues Sabino

**MOGI MIRIM - SP**

**2018**

**AGRADECIMENTOS**

Agradecemos a todos os professores do curso de análise e desenvolvimento de sistemas, pois cada um deles foi importante para concluir cada parte do projeto, dedicatória especial para os que nos acompanharam dia a dia na montagem do TG, tanto na parte escrita quanto na parte prática. Também agradecemos os nossos colegas de turma que sempre mantiveram um bom convívio, a todos que estão no grupo de TG, e ao nosso orientador que foi sempre prestativo e nos auxiliou nessa jornada.

**RESUMO**

Na atualidade, o tema “acessibilidade” vem tendo grande importância, pois todas as pessoas devem ter acesso a informação, mesmo que para isso, tenha de haver uma adaptação.

Utilizando de tecnologias como *machine learning*, e por meio de APIs, nosso trabalho propõe o desenvolvimento de um aplicativo mobile para pessoas que possuem deficiência visual.

Mesmo a demanda sendo baixa, entendemos a necessidade de um aplicativo desse tipo no mercado, atendendo pessoas cegas e com dificuldades de leitura.

O aplicativo consiste em transcrever imagens em textos, e, após essa transcrição, converter estes textos em áudio, fornecendo assim, a informação que a pessoa com deficiência necessita.

**Palavras-chaves**: *Machine learning*, engenharia, software, acessibilidade, metodologia.

**ABSTRACT**

Nowadays, the theme “Accessibility” is taking a great importance, because all people have the right to access every information available.

Using tecnologies like machine learning by APIs, our project propose the development of a mobile application for people that has visual impairment,

Even the demand being low, we understand the need of this kind of application in the market, helping blind people, and people that have reading difficulties.

The application works transcribing images in texts, and, after the transcription, it converts the texts in audios, providing the information that the person need.

**Key words**: Machine learning, engineering, software, accessibilty, methodology.

**LISTA DE FIGURAS**

[**Figura 1** - Blocos básicos da essência IoT 16](#_Toc516169561)

[**Figura 2** - Esquema geral para algoritmos de reconhecimento de imagens. 18](#_Toc516169562)

[**Figura 3** - Representação de uma imagem digital bidimensional monocromática. 20](#_Toc516169563)

[**Figura 4** - Representação de uma imagem digital bidimensional monocromática. 21](#_Toc516169564)

[**Figura 5** - Representação de uma imagem digital bidimensional monocromática. 22](#_Toc516169565)

[**Figura 6** - Componentes das cores de um pixel. 23](#_Toc516169566)

[**Figura 7** - Composição das bandas R, G e B. 23](#_Toc516169567)

[**Figura 8** - Camadas da Engenharia de Software 24](#_Toc516169568)

[**Figura 9** - Modelo Cascata 27](#_Toc516169569)

[**Figura 10** - Modelo Incremental 28](#_Toc516169570)

[**Figura 11** - Exemplo de código PHP (alto nível) 39](file:///C:\Users\Admin\Documents\GitHub\TG\Documents\TG_Vision.docx#_Toc516169571)

[**Figura 12** - Gráfico de linguagens de programação mais utilizadas. 40](#_Toc516169572)

[**Figura 13** - Arquitetura da plataforma Android 43](file:///C:\Users\Admin\Documents\GitHub\TG\Documents\TG_Vision.docx#_Toc516169573)

[**Figura 14** - Android Studio 46](#_Toc516169574)

**LISTA DE TABELAS**

[**Tabela 1** - Resumo das etapas de um sistema de processamento de imagens. 19](#_Toc516169624)

[**Tabela 2** - Atributos essenciais de um bom software 25](#_Toc516169625)

[**Tabela 3** - Ranking de popularidade das linguagens de programação. 39](#_Toc516169626)

[**Tabela 4** - Cronograma 52](#_Toc516169627)

**LISTA DE ABREVIATURAS E SIGLAS**

API - *Application Programming Interface*

DVC - *Distributed Version Control*

GCV – *Google Cloud Vision*

IDE - *Integrated Development Environment*

IEEE – *Institute of Electrical and Electronic Engineers*

IoT – *Internet of Things*

JVM - *Java Virtual Machine*

RFID – *Radio Frequency Identification*

SCM - *Source Control Management*

SO - Sistema Operacional

TTS – *Text To Speech*

Sumário

[1. Introdução 10](#_Toc516301400)

[2. Referencial Teórico 12](#_Toc516301401)

[2.1 Sistemas de Informação 12](#_Toc516301402)

[2.1.1 Machine Learning 12](#_Toc516301403)

[2.1.2 Tipos de Machine Learning 14](#_Toc516301404)

[2.1.3 Diferenças entre Inteligência Artificial e Machine Learning 15](#_Toc516301405)

[2.1.4 Internet das coisas 15](#_Toc516301406)

[2.1.5 Reconhecimento de Padrões em Imagens 17](#_Toc516301407)

[2.1.6 Modelos de uma Imagem digital 19](#_Toc516301408)

[2.2 Engenharia de Software 23](#_Toc516301409)

[2.2.1 Processo de Software 25](#_Toc516301410)

[2.2.2 Modelo de processo prescritivo 26](#_Toc516301411)

[2.2.2.1 Modelo cascata 27](#_Toc516301412)

[2.2.2.2 Modelo incremental 28](#_Toc516301413)

[2.2.3 Metodologia ágil 29](#_Toc516301414)

[2.2.4 Design Thinking 31](#_Toc516301415)

[2.2.4.1 Imersão 32](#_Toc516301416)

[2.2.4.2 Ideação 33](#_Toc516301417)

[2.2.4.3 Prototipação 34](#_Toc516301418)

[2.3 UML 35](#_Toc516301419)

[2.3.1 Diagrama de atividades 35](#_Toc516301420)

[2.3.2 Diagrama de caso de uso 36](#_Toc516301421)

[2.3.3 Diagrama de classes 37](#_Toc516301422)

[2.4 Linguagens de Programação 37](#_Toc516301423)

[2.4.1 Linguagens Web 40](#_Toc516301424)

[2.4.2 Linguagens Mobile 41](#_Toc516301425)

[2.4.3 Java 41](#_Toc516301426)

[2.4.4 Android 42](#_Toc516301427)

[2.5 Ferramentas de Desenvolvimento 44](#_Toc516301428)

[2.6 API 47](#_Toc516301429)

[2.6.1 Google CloudVision 48](#_Toc516301430)

[2.6.2 Google Cloud Text-to-Speech 48](#_Toc516301431)

[2.6.3 Google App Engine 49](#_Toc516301432)

[4. Análise e Desenvolvimento 50](#_Toc516301433)

[4.1 Público alvo 50](#_Toc516301434)

[4.2 Ouvindo o público alvo 50](#_Toc516301435)

[4.3 Ideação 50](#_Toc516301436)

[4.4 Gerenciamento do projeto 52](#_Toc516301437)

[5. conclusão 54](#_Toc516301438)

[Referências Bibliográficas 55](#_Toc516301439)

[APÊNDICE A – Documento de requisitos 60](#_Toc516301440)

[APÊNDICE B – Questões para entrevista com pessoas com deficiência visual 61](#_Toc516301441)

[APÊNDICE C – Diagramas 62](#_Toc516301442)

[ANEXO D – Respostas de um dos entrevistados 63](#_Toc516301443)

# Introdução

O universo das imagens divide-se no domínio de representações visuais de objetos materiais que representam o nosso meio ambiente visual e no domínio imaterial nas quais estas surgem como imaginações, fantasias, visões, modelos, ou seja, como representações mentais (JOLY, 2007; NÖTH & SANTAELLA, 2015).

É da natureza do ser humano fazer o reconhecimento de imagens através de padrões, sendo possível notar esse processo desde muito cedo, como no caso de um bebê que identifica a sua mãe (CARLO et All, 2012). Em alguns campos da neurociência, estuda-se como o cérebro reconhece e interpreta objetos. Segundo Charles E. Connor, diretor do *Zanvyl Krieger Mind/Brain Institute* da Universidade Johns Hopkins, “A visão não acontece no olho, ocorre em fases de processamento múltiplo no cérebro.” (ROSEN, 2018; NMLS, 2018).

Com o constante avanço tecnológico, grandes centros de pesquisas estão utilizando o *machine learning*, para ensinar as máquinas a visualizar, reconhecer, interpretar e tomar decisões. Mapeando-se algumas dentre as muitas necessidades enfrentadas por pessoas com deficiência visual, pode-se citar a dificuldade para realizar-se tarefas simples como reconhecer cenários comuns, desviar de obstáculos e fazer leituras básicas. Como a tecnologia *machine learning* pode contribuir para minimizar tais dificuldades? Como possibilitar um outro tipo de “visão” às pessoas com dificuldades visuais?

O objetivo principal do trabalho é a prototipação de um acessório ocular que sinaliza obstáculos próximos ao usuário e identifica padrões de letras e palavras informando através de áudio aquilo que está sendo visualizado através de uma conversão de texto em voz. Para isso, será realizada a integração entre tecnologia de reconhecimento de imagens com base na biblioteca *Google Cloud Vision*, a qual é disponibilizada de maneira gratuita com base em sua utilização, podendo ser aplicada para o uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de visão computacional.

Este trabalho será composto por 6 Capítulos. Inicialmente na Introdução, define-se o objeto de pesquisa proporcionando ao leitor uma ideia das necessidades, objetivos e expectativas da pesquisa. Na sequência apresenta-se a Revisão Teórica, a qual trará ferramentas que proporcionará uma fundamentação e validação da pesquisa. O terceiro capítulo apresenta a Análise e Desenvolvimento no qual dissertará sobre a modelagem, requisitos, materiais, métodos e análises dos resultados obtidos. No capítulo de Discussão, serão salientados os resultados obtidos buscando responder os questionamentos iniciais, validar as hipóteses propostas, cumprimento dos objetivos, mostrar as dificuldades encontradas e propor trabalhos futuros. Prosseguindo, apresenta-se a Bibliografia utilizada para o desenvolvimento e pesquisa. Por fim, os Apêndices trazem os documentos extras elaborados pelo grupo que complementam e exemplificam argumentações no texto, enquanto os Anexos exibem os documentos que não foram criados pelo grupo, mas que contribuíram para o desenvolvimento e embasamento acadêmico.

# Referencial Teórico

Este capítulo abordará todas as tecnologias, processos e procedimentos que fazem parte do desenvolvimento e elaboração do projeto. O projeto terá como base obras de autores especialistas em suas áreas, afim de que todo o conteúdo presente no trabalho tenha um embasamento robusto.

Logo de início, o primeiro tema que abordaremos será a tecnologia utilizada como base para este trabalho, o *Machine Learning.*

## Sistemas de Informação

Um sistema de informação é observado em qualquer processo, seja ele manual ou automatizado, abrangendo pessoas, máquinas e/ou métodos organizados para a distribuição da informação. Na área de sistemas destacam-se os *softwares* de aplicação, os quais são todos os tipos de programas utilizados na execução de tarefas específicas. Como exemplos disso, podem ser citados o *Microsoft Word* que serve para trabalhar com diversos tipos de textos, *Microsoft Excel* para fazer planilhas, gráficos e existem os *browsers* ou navegadores, sendo *softwares* utilizados para navegar na *Web*. Uma aplicação é desenvolvida em diversas linguagens de programação, em bancos de dados desenvolvidos, ou em qualquer outro recurso tecnológico que permita o desenvolvimento. Para criar uma aplicação deve-se ter um certo conhecimento na área de programação, seja ela em Delphi, C#, JAVA, Visual Basic, PHP (*Hypertext Processor*), entre várias outras. As aplicações são utilizadas frequentemente no dia a dia de todos, em um *smartphone* as aplicações são encontradas em qualquer função que queira ser exercida, nas empresas as aplicações são bem maiores e bem mais complexas, exercem uma responsabilidade maior para se ter acesso a elas. Para LAUDON, C., LAUDON, P.(2001, p.17):

Um sistema de informação pode ser definido tecnicamente como um conjunto de componentes inter-relacionados que coletam (ou recuperam), processam, armazenam e distribuem informações destinadas a apoiar a tomada de decisões, a coordenação e o controle de uma organização.

## 2.1.1 Machine Learning

Com o avanço da tecnologia, a computação pôde evoluir de uma maneira incrível. Computadores que outrora usavam válvulas, agora são feitos com circuitos integrados. Essas mudanças geravam questionamentos como: “Até onde a tecnologia pode chegar?”, ou até mesmo: “Os computadores substituirão os seres humanos?”.

Turing (1950) levantou o seguinte questionamento: “Poderiam as máquinas pensar?”.

Para responder a esse questionamento, primeiramente necessitamos entender o que é “pensar”. Quando pesquisamos nos dicionários, temos as definições: “Submeter (algo) ao processo de raciocínio lógico; exercer a capacidade de julgamento, dedução ou concepção”, “determinar pela reflexão”, “procurar lembrar-se, imaginar”, entre outras.

Turing (1950) cita que seria um absurdo que esse questionamento fosse feito com base em opiniões de pessoas apenas para a mera coleta de dados estatísticos, visto que abrange além da simples definição das palavras “máquina” e “aprender”.

Seguindo a linha de pensamento de Turing, Harnad (2008) propõe que o questionamento seja substituído, afim de termos a seguinte pergunta: “Poderiam as máquinas serem capazes de fazer o que nós, como entidades pensantes, somos capazes de fazer? E, se sim, como?”. Seguindo essa linha de raciocínio, temos a tecnologia *Machine Learning.*

*Machine learning* deriva do termo *“Learning Machines*” citado por Turing (1950), como máquinas que seriam capazes de aprender de maneira autômata, e foi cunhado por Samuel (1959).

Em outras palavras, podemos dizer que *Machine Learning* é o desenvolvimento de algoritmos com a capacidade de “aprender” sozinhos.

Mitchell (1997 p. 2, tradução nossa) define *Machine Learning* da seguinte maneira: “Pode-se dizer que um programa de computador aprende pela experiencia E respeitando uma classe de tarefas T e uma medida de performance P, se sua performance nas tarefas T, medidas por P, aumenta por meio da experiência E”[[1]](#footnote-2)

Em outras palavras, Mitchell (1997) exemplifica que um programa de computador que é desenvolvido para aprender a jogar damas melhorará sua performance ao ser medido por sua capacidade de vencer nas tarefas que envolvem o jogo, por meio da experiência obtida jogando contra si.

Sendo assim, temos T, como a tarefa: “Jogar damas”; P, como uma medida de performance para a porcentagem de jogos ganhos; e E, como a experiência obtida com as partidas jogadas contra si.

A cada nova partida, mais dados serão coletados para o aprendizado da máquina. Esses dados serão processados e irão prever jogadas que podem ser feitas. De maneira que quando a máquina jogar contra um humano, terá a informação de todas as possibilidades de jogadas existentes, conseguindo prever movimentos e estratégias.

Com evolução da tecnologia, o *Machine Learning* passou a ser mais difundido e utilizado, muitas vezes para reconhecimentos de padrões e aplicações para inteligência artificial.

Dentre as tecnologias que utilizar *Machine Learning*, podemos citar:

* Reconhecimento de imagens;
* Reconhecimento de voz;
* Reconhecimento facial;
* Aplicações para automatizar filtros de e-mail;
* Selecionar anúncios mais relevantes aos usuários;
* Informar as melhores rotas para desviar do trânsito;
* Recomendar filmes e músicas com base no gosto pessoal dos usuários;

entre diversas outras.

## 2.1.2 Tipos de Machine Learning

O Machine Learning pode ser categorizado através dos seguintes conceitos (LOPES, 2018):

* Supervisionado: Consiste apresentar a máquina dados de exemplo com respostas, afim de que possam ser mapeadas as entradas e saídas;
* Semi-Supervisionado: São apresentados dados incompletos a máquina;
* Aprendizado por reforço: Os dados são apresentados apenas como um retorno as ações da máquina em um ambiente dinâmico;
* Não supervisionado: A máquina fica responsável por agrupar e rotular os dados de maneira a encontrar padrões sem interferência externa.

## 2.1.3 Diferenças entre Inteligência Artificial e Machine Learning

O termo “Inteligência Artificial” foi evidenciado por John McCarthy no ano de 1956, usando a definição de máquinas que podem executar tarefas que são características da inteligência humana (MCCLELLAND, 2017).

McClelland (2017) cita que *Machine Learning* “é a maneira mais simples de se obter Inteligência Artificial” enquanto que Marr (2016), diz que o *Machine Learning* é o veículo que está guiando e dando velocidade ao desenvolvimento e avanço da Inteligência Artificial.

Em suas palavras, McClelland (2017) complementa que é possível obter inteligência artificial sem *Machine Learning*, mas que para isso, seria necessário programar linhas e mais linhas de código com regras complexas.

McClelland (2017) também menciona que o *Machine learning* auxiliou em um grande salto da Inteligência Artificial, devido a grande quantidade de dados que são coletados e trabalhados pelos bilhões de sensores que injetam a informação na internet. Todo esse amontoado de dados possibilita o avanço e melhora da Inteligência Artificial, e cria um ciclo, fazendo com que a Inteligência Artificial torne a internet das coisas útil.

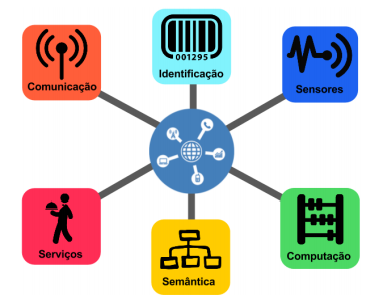
## 2.1.4 Internet das coisas

Segundo Gruman (2016), o conceito da Internet das Coisas (*Internet of Things* ou IoT) refere-se a um ambiente que engloba e conecta informações de diversos dispositivos como por exemplo smartphones, óculos, relógios, fones de ouvido, consoles de jogos, webcams, computadores, veículos, luzes da casa, e aplicações como compras, reabastecimento, avisos, dentre outros. Neste recente cenário, a variedade é crescente e conjectura-se que mais de 40 bilhões de dispositivos estarão conectados até 2020 (FORBES, 2014).

Além da conectividade de rede com ou sem fio, propõe-se como unidade básica de hardware, à concepção IoT, ao menos uma das seguintes características (LOUREIRO et al., 2003):

* unidade(s) de processamento;
* unidade(s) de memória;
* unidade(s) de comunicação e;
* unidade(s) de sensor(es) ou atuador(es).

A figura a seguir apresenta uma forma esquemática para a essência do IoT.

**Figura 1** - Blocos básicos da essência IoT

**Fonte:** Santos et Al (2016).

Os blocos apresentados na Figura 1 podem ser resumidos em (GOUSSEVSKAIA, 2017):

* **Identificação:** forma de identificar os objetos unicamente para conectá-los à Internet. Tecnologias como RFID, NFC (*Near Field Communication*) e endereçamento IP podem ser empregados para identificar os objetos.
* **Sensores/Atuadores:** os sensores possuem a função de coletar informações e, em seguida, encaminhar ou armazenar esses dados para *data warehouse*, *clouds* ou centros de armazenamento. Atuadores podem manipular o ambiente ou reagir de acordo com os dados lidos.
* **Comunicação:** refere-se às muitas técnicas utilizadas para conectar objetos inteligentes, além da função da atenção no consumo de energia dos objetos. Algumas das tecnologias usadas são *WiFi*, *Bluetooth*, IEEE 802.15.4 e RFID.
* **Computação:** abarca unidades de processamento como microcontroladores, processadores e FPGAs, responsáveis por executar algoritmos locais nos objetos inteligentes.
* **Serviços:** a IoT pode atender inúmeras classes de serviços, como por exemplo: Serviços de Identificação (temperatura, imagem, coordenadas geográficas, dentre outros); Serviços de Agregação de Dados que coletam e sumarizam dados homogêneos/heterogêneos obtidos dos objetos inteligentes; Serviços de Colaboração e Inteligência que agem sobre os serviços de agregação de dados para tomar decisões e reagir de modo adequado a um determinado cenário; e Serviços de Ubiquidade que visam prover serviços de colaboração e inteligência em qualquer momento e qualquer lugar em que eles sejam necessários.
* **Semântica:** diz respeito à competência de extração de conhecimento dos objetos na IoT. Trata da descoberta de conhecimento e uso eficiente dos recursos existentes na IoT, a partir dos dados existentes, com o objetivo de prover determinado serviço.

Mas qual o propósito de dispor de uma grande diversidade destes dispositivos conectados?

A sua versatilidade, possibilita estabelecer comunicação entre e com outros dispositivos, tornando a IoT deslumbrante. Para Goussevskaia (2017), com a IoT é possível controlar, trocar informações entre os dispositivos, acessar serviços da Internet e interagir com pessoas possibilitando com isso criar cidades inteligentes (*SmartCities*), saúde (*Healthcare*), casas inteligentes (*SmartHome*).

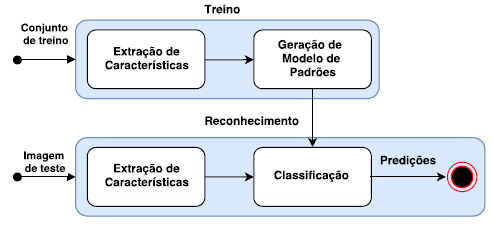
## 2.1.5 Reconhecimento de Padrões em Imagens

Para Lucena, Veloso e Lopes (2016), Queiroz e Gomes (2001), as dificuldades e os desafios no processamento de dados para o reconhecimento de objetos em imagens devem-se aos deveres dos sistemas serem eficientes no quesito taxas de reconhecimento, invariância das imagens quanto à rotação, ruído, luz, dentre muitas outras variáveis.

Usualmente as técnicas de processamento de imagens baseiam-se em métodos matemáticos que permitem descrever quantitativamente os mais diversos tipos de imagens. Para descrever-se uma imagem deve-se levar em consideração parâmetros que tenham característica bidimensional ou topológico, cor, luminosidade, dentre outros (ALBUQUERQUE; ALBUQUERQUE, 2004).

As principais tarefas de um software que identifica imagens sustentam-se na identificação de um objeto individual como um elemento de uma classe particular que engloba objetos com características semelhantes, necessitando para isso de descritores e classificadores. Pode-se resumir este processo nas fases de treinamento e reconhecimento, sendo que cada uma destas possuirá as etapas de extração de características e classificação (LUCENA; VELOSO; LOPES, 2016).

**Figura 2** - Esquema geral para algoritmos de reconhecimento de imagens.



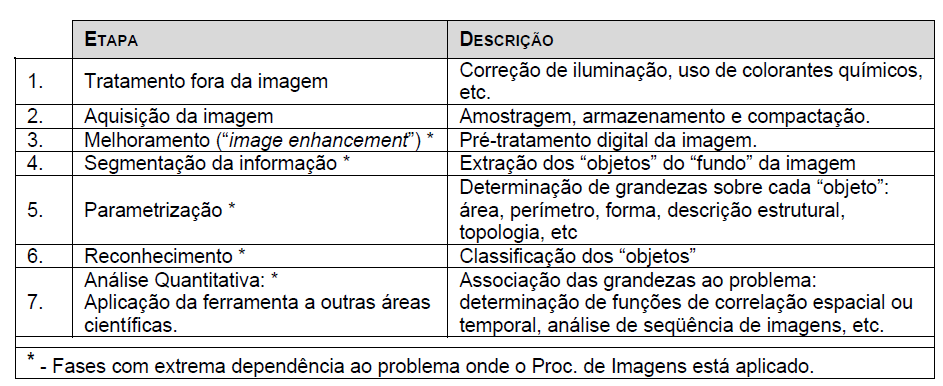
**Fonte:** (LUCENA; VELOSO; LOPES, 2016).

A fase de treinamento do algoritmo consiste na busca de padrões para representar os objetos de interesse. Os padrões são discriminados por um conjunto de características extraídas utilizando-se um descritor que pode agir sob toda a imagem ou ao redor de pontos de interesse (GONZALEZ; WOODS, 2006).

Basicamente, obtêm-se um conjunto de imagens para treino e cria-se um o descritor através de medidas de superfície, perímetros, comprimentos, espessura, posição, luminosidade, cor, dentre outros (ALBUQUERQUE; ALBUQUERQUE, 2004). As características extraídas são submetidas a um algoritmo de aprendizado de máquina que busca separar as características originadas dos descritores, diferenciando o objeto de interesse dos demais e assim produzindo um modelo dos padrões (ADDICAM; MALIK; TIAN, 2012).

Na fase de reconhecimento, o objetivo é identificar objetos na cena a partir de um conjunto de medições chamados vetores de características (descritores) extraídos de objetos padrões. A classificação é realizada através de uma medida de similaridade entre os descritores da imagem analisada e de todos os modelos de descritores que descrevem os objetos das classes aprendidas. Desta forma, um conjunto de objetos similares, com uma ou mais características semelhantes, é considerado como pertencente à mesma classe de padrões (GONZALEZ; WOODS, 2006) e (LUCENA; VELOSO; LOPES, 2016).

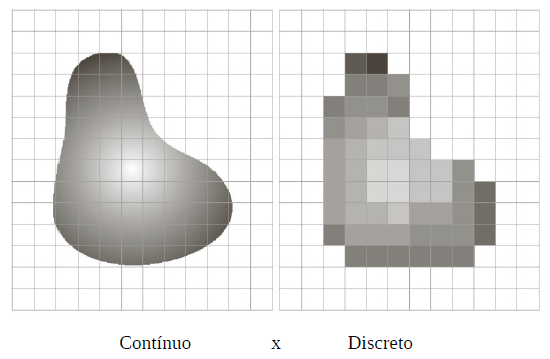
Detalhando um pouco mais as etapas de um sistema de processamento de imagens, Albuquerque e Albuquerque (2004) apresentam a tabela 1

**Tabela 1** - Resumo das etapas de um sistema de processamento de imagens.

**Fonte:** (MOREIRA, 2000).

## 2.1.6 Modelos de uma Imagem digital

Uma imagem digital pode ser classificada em monocromática ou colorida, sendo que em ambas as situações deve-se aproximar uma situação contínua da imagem real em um caso discreto no qual o computador será capaz de receber, processar e apresentar resultados. A figura 3 apresenta uma imagem monocromática bidimensional contínua e uma possível representação desta na forma de uma imagem digital bidimensional monocromática discreta.

**Figura 3** - Representação de uma imagem digital bidimensional monocromática.

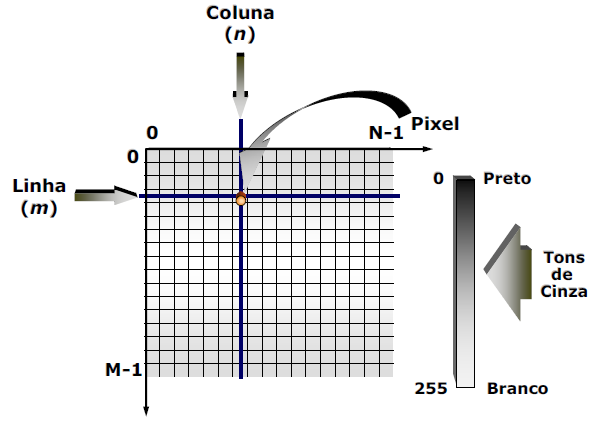
**Fonte:** (JACQUES JUNIOR, 2014)

Uma imagem monocromática é uma função bidimensional contínua , na qual as variáveis e são coordenadas definidas como espaciais e o valor de em qualquer ponto do domínio é proporcional à intensidade luminosa com características como brilho ou nível de cinza no ponto considerado (JÄHNE, 2002), (ACHARYA; RAY, 2005), ( RUSS, 2000) e (FORSYTH; PONCE, 2001).

A discretização da imagem é dada por matrizes nas quais imagens serão representadas como arranjos bidimensionais compostas por pontos, os quais serão denominados elemento de imagem ou pixel (QUEIROZ e GOMES, 2001).

A figura 4 apresenta um modelo, apresentado por Queiroz e Gomes (2001), da notação matricial usual para a localização de um pixel no arranjo de pixels de uma imagem bidimensional monocromático.

**Figura 4** - Representação de uma imagem digital bidimensional monocromática.



**Fonte:** (QUEIROZ e GOMES, 2001).

O índice denota a posição da linha onde se encontra o pixel enquanto o índice , denota a posição da coluna. Para uma imagem digital que contem linhas e colunas, devemos ter e . O referencial padrão utilizado para uma representação espacial de uma imagem digital pode ser visualizada na figura 2, na qual apresenta direções e , origem e os sentido de leitura (varredura) nos sentidos à direita de e para baixo de .

De uma forma geral, os tons de cinza são obtidos através de um dos valores no conjunto . No exemplo apresentado na figura 3, os tons de cinza estão associados ao conjunto , sendo o limite inferior () associado à um pixel preto e o limite superior () ao branco.

A intensidade luminosa no ponto pode ser decomposta em:

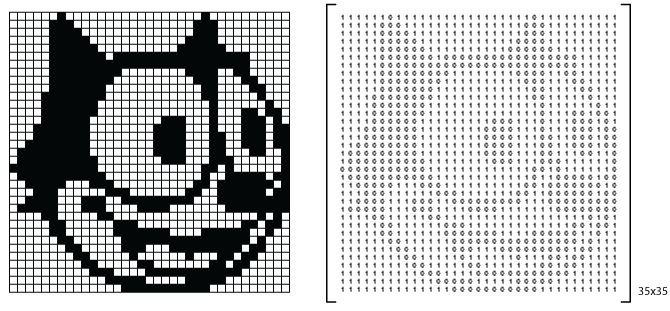
1. Componente de iluminação: Denotada por , está associada à quantidade de luz incidente sobre o ponto dependendo assim das características da fonte de iluminação.
2. Componente de reflectância: Denotada por , estando associada à quantidade de luz refletida pelo ponto dependendo assim das superfícies dos objetos.

É possível obter desta forma a função discreta

(1.0)

na qual e .

A figura 5 apresenta um modelo simples de uma imagem digital monocromática e a sua representação matricial.

**Figura 5** - Representação de uma imagem digital bidimensional monocromática.

**Fonte:** (PEREIRA; PESCO; BORTOLOSSI, 2006).

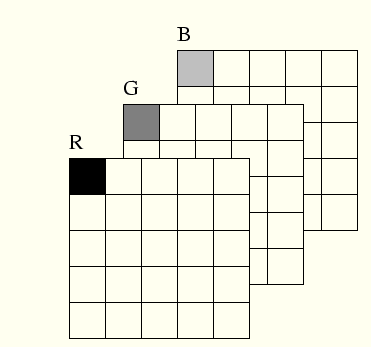
Para uma imagem digital colorida, pode-se utilizar o sistema RGB. Neste, um pixel será representado por um vetor cujas componentes representam as intensidades de vermelho, verde e azul de sua cor. Assim, associa-se a cada pixel três valores inteiros entre 0 e 255, cada qual indicando qual a intensidade de vermelho, verde e azul no ponto da imagem. Como exemplo, um pixel branco terá enquanto um pixel negro terá .

Queiroz e Gomes (2001), dizem que pode-se interpretar a imagem colorida como a composição de três imagens monocromáticas, ou seja:

(1.1)

na qual , , representam, respectivamente, as intensidades luminosas das componentes vermelha, verde e azul da imagem, no ponto . A imagem da figura 5, apresenta uma forma esquemática de como é realizada a combinação dada pela função (1.1).

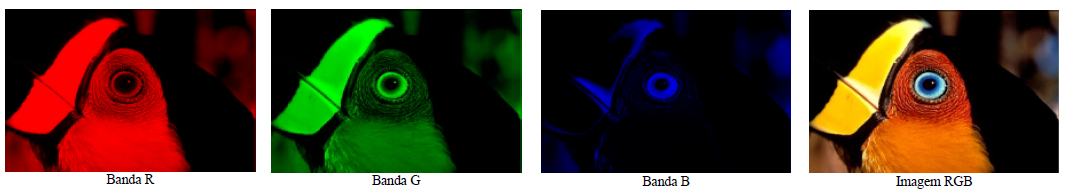
**Figura 6** - Componentes das cores de um pixel.



**Fonte:** (MOREIRA, 2000).

A imagem da figura 7, apresenta a representação da imagem de um tucano através das bandas R, G, B e da combinação RGB.

**Figura 7** - Composição das bandas R, G e B.



**Fonte:** (BATISTA, 2005).

## 2.2 Engenharia de Software

Para Pressman (2011) o sucesso de um *software* se dá quando o mesmo supre as necessidades dos usuários, fica funcional por um longo período, possui facilidade de manutenção e adaptação, e possui fácil usabilidade. Para alcançar tudo isso e evitar os pontos negativos, é necessário ter-se uma abordagem de engenharia nos quesitos projetar e construir do *software*.

A engenharia de *software* é um conjunto de métodos, processos e ferramentas utilizados para o desenvolvimento de um *software*, define Pressman (2011). Já Sommerville (2011 p. 4) define que a engenharia de *software*: "É uma disciplina de engenharia que se preocupa com todos os aspectos de produção de software. "

**Figura 8** - Camadas da Engenharia de Software



**Fonte:** (PRESSMAN; MAXIM, 2016, p. 16).

Para Sommerville (2011 p. 3):

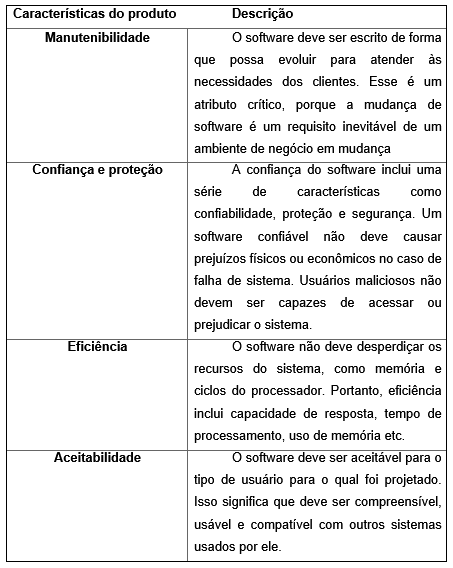
A engenharia de software tem por objetivo apoiar o desenvolvimento profissional de software, mais do que a programação individual. Ela inclui técnicas que apoiam especificação, projeto e evolução de programas, que normalmente não são relevantes para o desenvolvimento de software pessoal [...]

Muitas pessoas pensam que software é simplesmente outra palavra para programas de computador. No entanto, quando falamos de engenharia de software, não se trata apenas do programa em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse programa operar corretamente. [...]

Segundo Pressman (2011 p. 39) apud (BAUER, 1969):

A engenharia de software é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter o software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais.

Na tabela 1, apresenta-se os conjuntos de atributos de um bom *software* para Sommerville (2011):

**Tabela 2** - Atributos essenciais de um bom software

**Fonte:** (SOMMERVILLE, 2011).

## 2.2.1 Processo de Software

Sommerville (2011 p. 18) diz que "um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software". Este cita quatro atividades fundamentais para engenharia de software:

* **Especificação de software:** Define-se quais serão as funcionalidades e restrições do software
* **Projeto e implementação de software:** Projeta-se o desenvolvimento para atender às especificações
* **Validação de software:** Valida-se para que o *software* atenda todas as necessidades do cliente
* **Evolução de software:** Aperfeiçoa-se o software para que se adeque as necessidades de mudança do cliente

Ainda sobre processos, Sommerville (2011 p. 19) cita que:

Os processos de software são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software. Os processos têm evoluído de maneira a tirarem melhor proveito das capacidades das pessoas em uma organização, bem como das características específicas do sistema em desenvolvimento.

Para Pressman (2011) o processo na engenharia de *software* é adaptável, possibilitando que a equipe selecione e escolha o melhor conjunto de ações e tarefas. Ele ainda considera que uma abordagem de processos genérica na engenharia de *software* engloba: Comunicação (compreender os objetivos e levantar as necessidades), Planejamento (criar um mapa para guiar a equipe), Modelagem (realizar um esboço), Construção (produzir o código e realizar testes) e Emprego (entrega do projeto e recebimento do feedback por parte do cliente).

## 2.2.2 Modelo de processo prescritivo

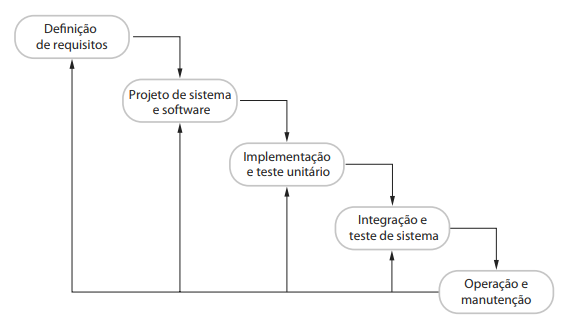
Os modelos de processo prescritivo, também conhecidos como modelos tradicionais surgiram com o intuito de organizar o caos existente em toda a parte de desenvolvimento de software, trazendo uma estrutura que pode ser utilizada várias vezes e um roteiro para a equipe seguir (PRESSMAN, 2011).

Para Sommerville (2011 p. 19):

[...] um modelo de processo de software é uma representação simplificada de um processo de software. Cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele.

## 2.2.2.1 Modelo cascata

O modelo cascata para Sommerville (2011 p. 19): "considera as atividades fundamentais do processo de especificação, desenvolvimento, validação e evolução, e representa cada uma delas como fases distintas". O autor cita que as fases desse modelo são:

**Figura 9** - Modelo Cascata

**Fonte:** (SOMMERVILLE, 2011)

Especificando com detalhes cada um dos blocos do modelo cascata, tem-se:

* **Análise e definição de requisitos:** É definido como o sistema deve ser, suas restrições e como funcionará;
* **Projeto de sistema e software:** Envolve a identificação e a descrição das abstrações fundamentais do software;
* **Implementação e teste unitário:** O projeto é desenvolvido em unidades separadas, e cada unidade possui testes para verificar se atende a sua especificação;
* **Integração e teste de sistema:** Após os testes unitários, as unidades que outrora estavam separadas, agora são integradas e é feito um teste completo do sistema;
* **Operação e manutenção:** O projeto é colocado em uso, e são feitas correções de erros não descobertos previamente e adaptações a medida que novos requisitos vão surgindo.

O modelo cascata opera de maneira sequencial e linear, iniciando-se com o levantamento das necessidades do cliente, planejamento do cronograma, modelagem do projeto, construção (codificação) e emprego (entrega e suporte) (PRESSMAN, 2011).

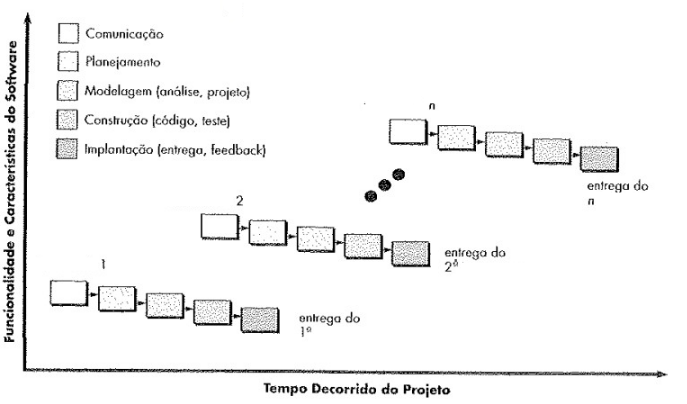
## 2.2.2.2 Modelo incremental

Pressman (2011), diz que este modelo segue sequências lineares, escalonando conforme o tempo avança. Cada sequência linear gera um incremento de software que vão moldando o projeto.

Para Sommerville (2011 p. 21):

O desenvolvimento incremental é baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido.

**Figura 10** - Modelo Incremental



**Fonte:** (PRESSMAN, 2011)

Sommerville (2011 p. 22):

Cada incremento ou versão do sistema incorpora alguma funcionalidade necessária para o cliente. Frequentemente, os incrementos iniciais incluem a funcionalidade mais importante ou mais urgente. Isso significa que o cliente pode avaliar o sistema em um estágio relativamente inicial do desenvolvimento para ver se ele oferece o que foi requisitado. Em caso negativo, só o incremento que estiver em denvolvimento [sic] [desenvolvimento] no momento precisará ser alterado e, possivelmente, nova funcionalidade deverá ser definida para incrementos posteriores.

Além dos modelos citados, existem outros modelos para o desenvolvimento de *software* como o modelo espiral, modelo evolucionário, entre outros.

Para este trabalho, optou-se por utilizar um modelo de desenvolvimento orientado a um método ágil. No tópico a seguir, será apresentado os principais conceitos a respeito de metodologias ágeis, além da metodologia de desenvolvimento escolhida para este projeto.

## 2.2.3 Metodologia ágil

O desenvolvimento ágil teve seu conceito iniciado em meados de 1990. É uma metodologia que coloca o cliente em evidência e visa os seguintes pontos (MANIFESTO ÁGIL, 2001):

* **Indivíduos e interações** mais que processos e ferramentas;
* **Software em funcionamento** mais que documentação abrangente;
* **Colaboração com o cliente** mais que negociação de contratos;
* **Responder a mudanças** mais que seguir um plano.

O Manifesto Ágil (2001) é um conjunto de princípios, dentre eles:

* Priorizar a satisfação do cliente, através da entrega adiantada e contínua de software de valor.
* Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
* Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
* Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito

Segundo Sbrocco e Macedo (2012):

O manifesto ágil foi criado considerando a velocidade demandada para novos sistemas de software, com o objetivo de abandonar métodos antigos que se mostravam ultrapassados devido ao uso de hardwares mais avançados, linguagens de programação, ambientes de desenvolvimento e necessidades organizacionais. Trata-se de um documento que encoraja o uso de melhores métodos de desenvolver sistemas de software, contendo um conjunto de princípios que definem critérios para os processos de desenvolvimento ágil de sistemas.

Sommerville (2011) define os métodos ágeis da seguinte maneira:

Métodos ágeis, universalmente, baseiam-se em uma abordagem incremental para a especificação, o desenvolvimento e a entrega do software. Eles são mais adequados ao desenvolvimento de aplicativos nos quais os requisitos de sistema mudam rapidamente durante o processo de desenvolvimento. Destinam-se a entregar o software rapidamente aos clientes, em funcionamento, e estes podem, em seguida, propor alterações e novos requisitos a serem incluídos nas iterações posteriores do sistema. Têm como objetivo reduzir a burocracia do processo, evitando qualquer trabalho de valor duvidoso de longo prazo e qualquer documentação que provavelmente nunca será usada.

Para Pressman (2011) o fator humano é primordial em um desenvolvimento de software ágil, levando em consideração que o processo se constrói com base nas necessidades das pessoas e equipes, e não o contrário. Para isso, as pessoas devem possuir alguns traços-chave como: competência, foco comum, colaboração, habilidades na tomada de decisão, habilidade de solução de problemas confusos, confiança mútua, respeito e auto-organização.

Podemos comparar quais são as principais diferenças entre o desenvolvimento tradicional e o desenvolvimento ágil. Para Sbrocco e Macedo (2011 p. 184):

Existem vários aspectos responsáveis por diferenciar as metodologias tradicionais das ágeis, os quais podem ser observados em regras que explicam como produzir e conduzir processos, na forma peculiar de desenvolver software existente no ambiente de desenvolvimento e na maneira de pensar dos integrantes da equipe. Talvez esta última seja, de fato, a principal diferença que podemos observar, pois pensar diferente implica a adoção de princípios de desenvolvimento diferentes.

Outra característica importante das metodologias ágeis é que elas são preparadas para aceitar mudanças durante o processo de desenvolvimento de software, ao passo que as metodologias tradicionais são resistentes a elas. Metodologias ágeis são baseadas em dados estatísticos obtidos de históricos referentes à implementação do código. Já os métodos tradicionais utilizam como base normas que definem padrões a serem seguidos. A estratégia de trabalho adotada é imposta pela própria equipe de desenvolvimento, não exigindo um grande controle dos processos, o que é bem diferente do observado nas metodologias tradicionais, uma vez que percebemos claramente a imposição mais acirrada, por meio de inúmeras normas e políticas que devem ser respeitadas.

## 2.2.4 Design Thinking

O *Design Thinking* surge da ideia de utilizar os mesmos pensamentos e percepções de um *designer.*

Para Viana et. al. (2012 p. 13):

O designer enxerga como um problema tudo aquilo que prejudica ou impede a experiência (emocional, cognitiva, estética) e o bem-estar na vida das pessoas (considerando todos os aspectos da vida, como trabalho, lazer, relacionamentos, cultura etc.). Isso faz com que sua principal tarefa seja identificar problemas e gerar soluções.

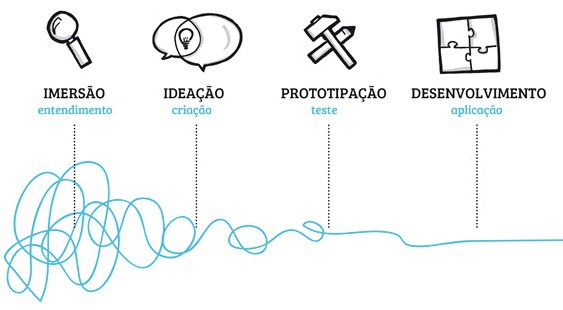
O *designer*, segundo Viana et. al. (2012) tem a percepção de que problemas possuem naturezas diversas e devem ser lidados conforme o ambiente, contexto, experiências pessoais e os processos que ocorrem na vida dos indivíduos. Mapeando estes pontos, é possível identificar as barreiras e dificuldades, e, tendo esse conhecimento, é possível ser mais assertivo na solução do cenário.

O *design thinking* utiliza uma maneira de pensar “fora da caixa”, o pensamento abdutivo. Segundo Viana et. al (2012 p. 13):

Nesse tipo de pensamento, busca-se formular questionamentos através da apreensão ou compreensão dos fenômenos, ou seja, são formuladas perguntas a serem respondidas a partir das informações coletadas durante a observação do universo que permeia o problema. Assim, ao pensar de maneira abdutiva, a solução não é derivada do problema: ela se encaixa nele.

Nos próximos tópicos serão abordados os processos desta metodologia, que são: Imersão, Ideação e Prototipação. Na figura 5 observa-se os processos não lineares do *Design Thinking:*

**Figura 5:** Estrutura do Processo *Design Thinking.*



**Fonte:** (MODIFICADO SEBRAE, 2017).

## 2.2.4.1 Imersão

O processo de imersão consiste em aproximar-se do ponto de vista do cliente/usuário, podendo ser dividido em imersão preliminar e imersão profunda.

**Imersão preliminar**

A imersão preliminar para Viana et. al. (2012 p. 22): “[...] tem como finalidade definir o escopo do projeto e suas fronteiras, além de identificar os perfis de usuários e outros atores-chave que deverão ser abordados.”

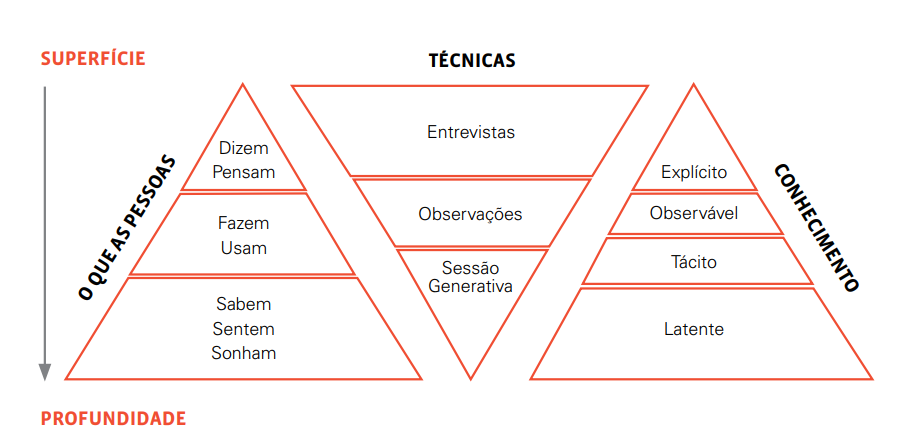
Essa etapa começa com um processo de Reenquadramento no qual a equipe de projeto reúne-se com os profissionais da empresa contratante, seja em entrevistas individuais ou em dinâmicas coletivas, para olhar o problema sob outras perspectivas e definir as fronteiras do projeto. Além disso, a equipe de projeto costuma realizar uma Pesquisa Exploratória em campo para ouvir sobre o tema de forma a criar um entendimento inicial dos usuários e atores envolvidos no contexto e auxiliar na definição dos perfis principais para serem investigados a seguir, na Pesquisa em Pronfundidade [sic] [profundidade]. Assim como se realiza uma Pesquisa Desk na busca por tendências sobre o assunto no Brasil e exterior. (VIANA et. al., 2012, p. 24)

**Imersão em profundidade**

Esta imersão consiste em mergulhar no ambiente do usuário/cliente, identificando suas necessidades e expectativas. Segundo Viana et. al (2012), nesta etapa é possível utilizar técnicas como entrevistas e observação, para mergulhar no contexto do que está sendo explorado no projeto.

A ideia é identificar comportamentos extremos e mapear seus padrões e necessidades latentes. A pesquisa é qualitativa e não pretende esgotar o conhecimento sobre segmentos de consumo e comportamento, mas ao levantar oportunidades de perfis extremos, permite que soluções específicas sejam criadas. Soluções essas que muitas vezes atendem a mais grupos, mas que não teriam surgido se o olhar não tivesse sido direcionado para as diferenças. Para tal, os membros da equipe de projeto vão ao encontro do cliente/usuário do produto ou serviço em questão, para observar ou interagir com ele no contexto de uso de forma a aproximar-se de seus pontos de vista e descobrir não só o que falam, mas também o que/como fazem e sentem. (VIANA et. al., 2012, p. 36)

**Figura 6** - Imersões



**Fonte:** Viana et. al. (2012)

**Análise e síntese**

Após a conclusão das duas etapas anteriores, toda a massa de dados e informações são organizadas afim de gerarem padrões para auxiliarem na compreensão do problema.

## 2.2.4.2 Ideação

Nesta fase, a criatividade é o ponto chave para desenvolver ideias e soluções para os problemas no contexto apresentado. Segundo Viana et.al. (2012), também são utilizadas as ferramentas do processo de análise para ter sempre a síntese dos dados.

Para Viana et. al. (2012 p. 101): “A fase de ideação geralmente se inicia com a equipe de projeto realizando Brainstormings (uma das técnicas de geração de ideias mais conhecidas) ao redor do tema a ser explorado e com base nas ferramentas.”

**Brainstorming**

O *brainstorming*, geralmente executado em grupo,consiste na geração do maior número de ideias no menor tempo possível. Os participantes ficam à vontade, mas evitando perder o foco do contexto do problema a ser solucionado.

Segundo Viana et. al. (2012 p. 101): “A qualidade e a assertividade das ideias geradas se atinge através da quantidade. Quanto maior a quantia de ideias geradas pela equipe, maior é a chance de produzir uma solução inovadora e funcional.”

Ideias ousadas são bem-vindas, pois podem levar a soluções inovadoras. Críticas a essas ideias, não devem atrapalhar este processo. Dessa maneira, a avaliação fica para um momento posterior.

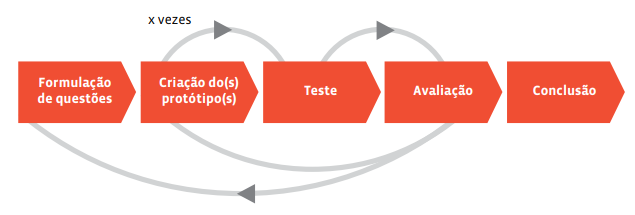
## 2.2.4.3 Prototipação

No processo de prototipação, todas as ideias, outrora abstratas, agora são representadas fisicamente. Essa representatividade, se dá sob dois aspectos:

* **Da ótica da equipe de projeto:** a equipe da mais detalhes a forma física da ideia, aumentando sua fidelidade conforme o processo avança.
* **Do ponto de vista do usuário:** com bases em suas interações, os usuários podem fornecer avaliações e ideias para o avanço do protótipo.

Para Viana et. al. (2012 p. 124): “Protótipos reduzem as incertezas do projeto, pois são uma forma ágil de abandonar alternativas que não são bem recebidas e, portanto, auxiliam na identificação de uma solução final mais assertiva.”

**Figura 7-** Prototipação



**Fonte:** Viana et. al. (2012)

Esses protótipos podem variar desde uma simples representação da ideia em uma folha de papel (Baixo nível), até algo que seja representado de maneira que chegue próximo da solução final (Alto Nível).

O contexto nesses cenários também é importante, segundo Viana et. al. (2012 p. 123):

O teste de um protótipo pode envolver ou não usuários finais e ser realizado desde em um laboratório, até no ambiente final onde o produto ou serviço será usado. As diferentes combinações desses elementos representam os níveis de contextualidade.

## 2.3 UML

UML é um acrônimo para a expressão *“Unified Modeling Language”*, ou, em português, linguagem de modelagem unificada. É uma notação gráfica com aspectos do sistema a ser desenvolvido, atuando geralmente em projetos utilizando da prática de orientação a objeto.

Segundo Sommerville (2011 p. 82):

Os modelos são usados durante o processo de engenharia de requisitos para ajudar a extrair os requisitos do sistema; durante o processo de projeto, são usados para descrever o sistema para os engenheiros que o implementam; e, após isso, são usados para documentar a estrutura e a operação do sistema.

Para essas notações, a UML dispõe de diagramas que possuem variedade de uso, desde de documentar ações do usuário no sistema, até diagramas sobre classes que o projeto deverá ter no código.

Para este projeto, serão utilizados os diagramas de atividades, caso de uso e de classe.

A seguir falaremos destes diagramas.

## 2.3.1 Diagrama de atividades

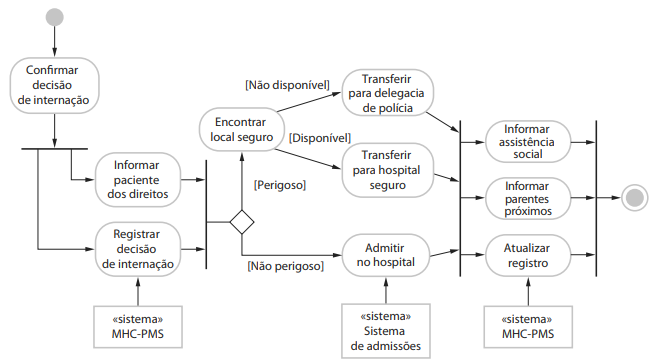
Segundo Sommerville (2011):

Os diagramas de atividades são destinados a mostrar as atividades que compõem um processo de sistema e o fluxo de controle de uma atividade para a outra. O início de um processo é indicado por um círculo preenchido; o fim, por um círculo preenchido dentro de outro círculo. Os retângulos com cantos arredondados representam atividades, ou seja, os subprocessos específicos que devem ser realizados.

Neste diagrama, as setas representam o fluxo de trabalho de uma atividade. Barras sólidas são utilizadas como coordenação das atividades. Quando mais de uma seta se encontra em uma barra sólida, significa que é necessário que as atividades sejam concluídas antes de passarem para a próxima atividade. Quando ocorre o contrário, significa que as atividades podem ser executadas em paralelo.

Além disso, também existem pontos de decisão que consistem em dar as atividades, um fluxo de trabalho diferente com base na condição executada.

**Figura 8** – Exemplo Diagrama de Atividades



**Fonte:** Sommerville, (2011).

## 2.3.2 Diagrama de caso de uso

O diagrama de caso de uso é uma representação do que o usuário espera com base em suas interações com o sistema.

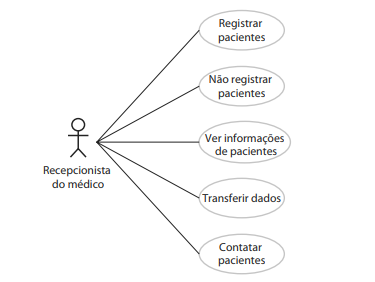
Para Sommerville (2011 p. 86):

Cada caso de uso representa uma tarefa discreta que envolve a interação externa com um sistema. Em sua forma mais simples, um caso de uso é mostrado como uma elipse, com os atores envolvidos representados por figuras-palito.

Sommerville (2011 p. 87) atenta para os seguintes pontos na utilização do diagrama:

Diagramas de casos de uso dão uma visão simples de uma interação. Logo, é necessário fornecer mais detalhes para entender o que está envolvido. Esses detalhes podem ser uma simples descrição textual, uma descrição estruturada em uma tabela ou um diagrama de sequência [...] Você deve escolher o formato mais adequado, dependendo do caso de uso, e o nível de detalhamento que você acredita ser necessário no modelo.

**Figura 9 -** Exemplo Diagrama de Caso de Uso



**Fonte:** Sommerville, (2011).

Cada caso de uso também pode possuir uma inclusão ou extensão. Na inclusão, sempre quando o usuário interage com um caso de uso, outro será executado. Podemos exemplificar que o sistema possui o caso de uso “Marcar Consulta” incluindo o caso de uso “Consultar disponibilidade”. Se o usuário interagir com o primeiro, o segundo também será executado, neste cenário, validando se existe disponibilidade para marcar a consulta na data informada.

No caso da extensão, o usuário pode interagir com um caso de uso vinculado ao que ele está interagindo no momento. Exemplificando, o usuário pode interagir com o caso de uso “Efetuar pagamento”, e ao interagir tem a possibilidade de efetuar um pagamento sem cadastro ou efetuando um cadastro. Se optar por efetuar o cadastro, podemos considerar este como uma extensão do caso de uso “Efetuar pagamento”.

## 2.3.3 Diagrama de classes

Sommerville (2011) define o diagrama de classe da seguinte maneira:

Os diagramas de classe são usados no desenvolvimento de um modelo de sistema orientado a objetos para mostrar as classes de um sistema e as associações entre essas classes. Em poucas palavras, uma classe de objeto pode ser pensada como uma definição geral de um tipo de objeto do sistema. Uma associação é um link entre classes que indica algum relacionamento entre essas classes. Consequentemente, cada classe pode precisar de algum conhecimento sobre sua classe associada.

O diagrama de classe realiza a abstração de coisas reais que serão implementadas no programa. Por exemplo, temos um paciente, este paciente deseja fazer o seu cadastro e marcar uma consulta. Sendo assim, temos a classe “Paciente” com os seus dados: “Nome, CPF, endereço, etc” podendo realizar as ações “Cadastrar” e “Agendar consulta”.

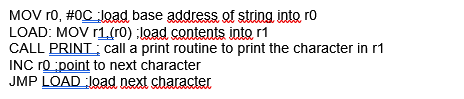
O nome da classe de objeto está na seção superior. Os atributos de classe estão na seção do meio. O que deve incluir os nomes dos atributos e, opcionalmente, seus tipos. As operações (chamadas métodos, em Java e em outras linguagens de programação OO) associadas à classe de objeto estão na seção inferior do retângulo.(SOMMERVILLE, 2011)

## 2.4 Linguagens de Programação

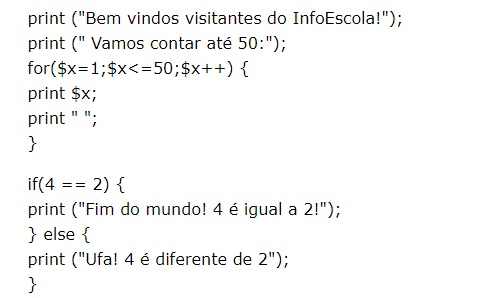
Um computador pode ser considerado uma “supercalculadora”, mas o que é preciso para ela realizar as operações? Simples, basta apenas você digitar os números nela, a linguagem de programa faz a mesma coisa com o computador, ela mostra para o computador o que deve ser feito e ele apenas realiza a ação solicitada pelo usuário, basicamente ele serve como uma ponte entre computadores e humanos.

Existem dois tipos de linguagens de programação: as de baixo nível e as de alto nível. Os computadores interpretam tudo como números em base binária, ou seja, só entendem zero e um. As linguagens de baixo nível são interpretadas diretamente pelo computador, tendo um resultado rápido, porém é muito difícil e incômodo se trabalhar com elas. Exemplos de linguagens de baixo nível são a linguagem binária e a linguagem Assembly.:

**Figura 10** - Exemplo de código em Assembly

Como pode-se notar, é uma linguagem bastante complicada. (ANDRADE, 2018)

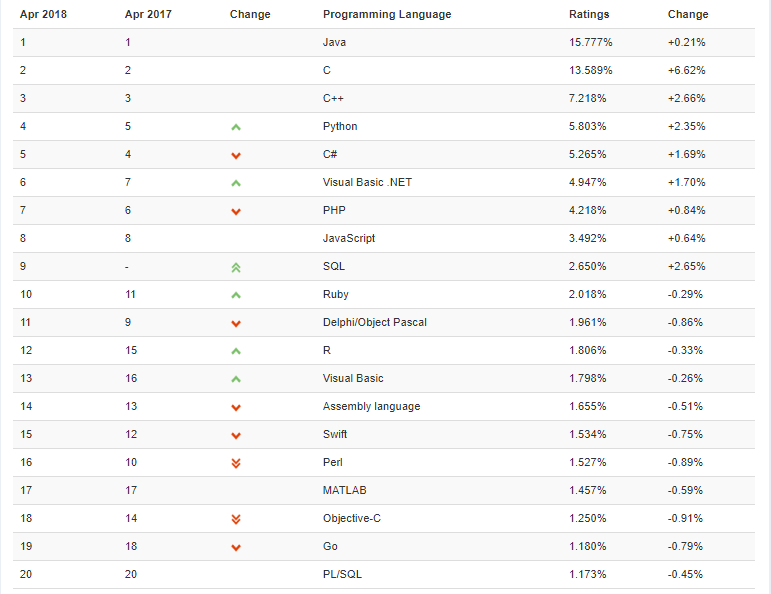
As linguagens de alto nível são bem mais fáceis de se trabalhar devido ao uso de palavras em inglês assimilando com as ações necessárias. Essas ações não são compreendidas pelo computador que só entendo código binário (0 e1), devido a isso usa-se um programa para compilar essas palavras para códigos, também chamado de compilador. Quando se está programando, o código é digitado com as palavras chaves contendo as ações que o computador tem que realizar é chamado de código-fonte, depois compila-se o arquivo para que ele se transforme em código-binário isso irá gerar um programa executável, isso é o que o computador vai interpretar como seu código.

Existem algumas linguagens de programação que não necessitam de compiladores, como o PHP, uma linguagem dedicada à produção de websites dinâmicos. As instruções em PHP são compiladas e executadas ao mesmo tempo. (ANDRADE,2018)

**Figura 11** - Exemplo de código PHP (alto nível)

Logo em sequência encontra-se a tabela 2 mostrando as linguagens de programação mais usadas no período de abril entre 2017 e 2018. A linguagem Java está em primeiro lugar com 15,777% de utilização de usuários, em segundo lugar vem o C com 13,589% e em terceiro vem o C++ com 7,218 %.

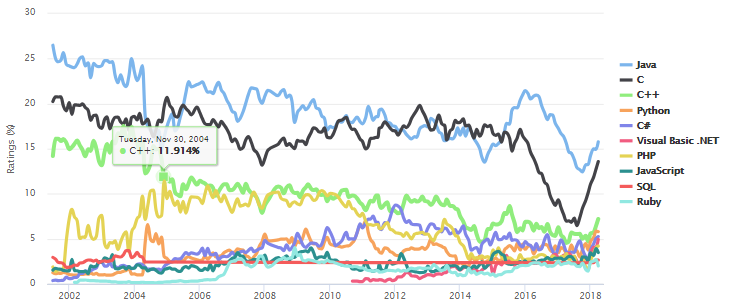
**Tabela 3** - Ranking de popularidade das linguagens de programação.



**Fonte:** (TIOBE, 2018)

A figura 12 apresenta um gráfico com o crescimento dos tipos de linguagens de programação mais utilizados no mundo. Observa-se que a linguagem Java em 2002 era a mais utilizada, mas veio perdendo espaço para a linguagem C, que em 2004 conseguiu ultrapassar o Java em utilização, mas dois anos depois continuou ficando atrás dele, até em 2012 novamente consegue passar a frente dele, assim permanecendo até 2018.

**Figura 12** - Gráfico de linguagens de programação mais utilizadas.

**Fonte:** TIOBE 2018

## 2.4.1 Linguagens Web

A área mais dinâmica para novas aplicações relacionadas a programação é a Internet, que é o meio onde existe uma gama de aplicações comerciais, acadêmicas, governamentais e industriais. As linguagens de programação que suportam a computação centrada na Web usam a programação orientada a evento, promovendo a interação sistema-usuário. Essas linguagens também usam o paradigma orientado a objetos, pelo simples fato que diversas entidades que aparecem na tela do usuário são modeladas de forma mais natural como objetos que enviam e recebem mensagens. Segundo Tucker, Noonan (2008, p.26):

A noção de computação centrada na Web e, consequentemente, de programação centrada na Web é motivada por um modelo interativo no qual um programa permanece ativo continuamente esperando pelo próximo evento, respondendo a este evento e retornando a seu estado continuamente ativo.

As linguagens de programação mais famosas que suportam computação centrada na Web incluem Perl, PHP, Java e Python.

## 2.4.2 Linguagens Mobile

O [desenvolvimento](https://www.oficinadanet.com.br/desenvolvimento) de aplicativos móveis já é muito mais que realidade no nosso cotidiano, dentre as plataformas móveis que se destacam temos o [Android](http://www.oficinadanet.com.br/artigo/outros_sistemas/google_android_os) da Google e o iOS da Apple, porém existem outras plataformas menos populares como o Windows Phone e a BlackBerry. Antes de desenvolver uma aplicação para esses tipos de dispositivos é fundamental que se tenha um conhecimento em relação as linguagens de programação que são compatíveis com esse modelo.

As principais linguagens de programação disponibilizam uma enorme base de códigos abertos, bibliotecas e frameworks para a comunidade de desenvolvedores, tornando assim o trabalho dos programadores mais fácil. Podemos destacar as linguagens Java, Phyton, Objective-C, Swift, C#. Segundo MONTEIRO (2015, p.2)

A plataforma Android desfruta hoje de um papel de destaque no mercado, tanto pela quantidade significativa de dispositivos produzidos como também por oferecer uma API rica, disponibilizando fácil acesso a vários recursos de hardware, tais como Wi-Fi e GPS, além de boas ferramentas para o desenvolvedor. A facilidade de desenvolver utilizando uma linguagem de programação (Java) bastante disseminada, a simplicidade e baixo custo para a publicação de aplicativos na loja Google Play e a quantidade de dispositivos Android em uso no mundo só fazem aumentar a popularidade da plataforma[...].

## 2.4.3 Java

Java é uma linguagem de programação multiplataforma, orientada a objetos, e tem a possibilidade de rodar em qualquer sistema operacional que possua um interpretador instalado. O interpretador é denominado como JVM, sendo um *software* que transforma o código da linguagem em comandos executáveis pelo sistema operacional.

A linguagem faz uso de um conceito diferente na geração dos códigos binários, ao invés de gerar códigos distintos para plataformas distintas, é gerado um binário que pode ser executado em qualquer plataforma, dentro do interpretador. Esse código “genérico” é conhecido como *bytecode*. Segundo Claro, Sobral (2008, p.15):

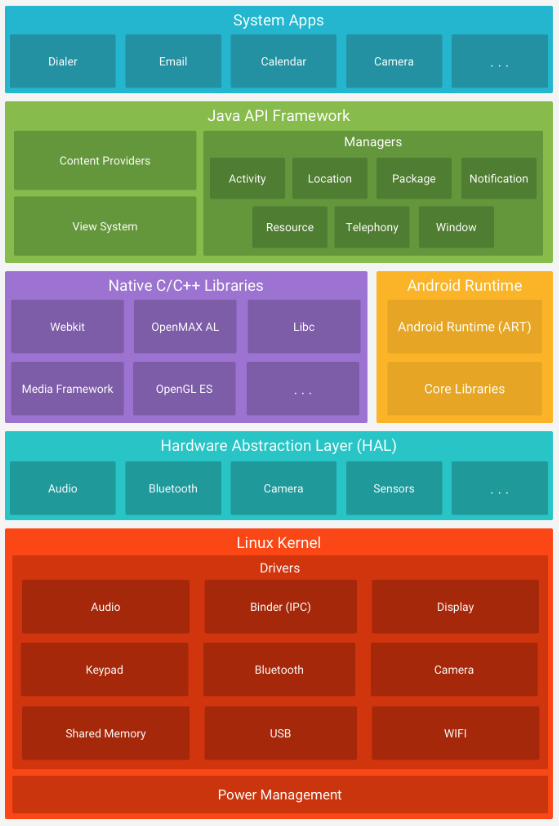
Um programa desenvolvido em Java necessita ser compilado, gerando um bytecode. Para executá-lo é necessário então, que um interpretador leia o código binário, o bytecode e repasse as instruções ao processador da máquina específica. Esse interpretador é conhecido como JVM (Java Virtual Machine). Os bytecodes são conjuntos de instruções, parecidas com código de máquina. É um formato próprio do Java para a representação das instruções no código compilado.

Pelo fato de ser uma linguagem que não depende de sistema operacional, a mesma pode ser utilizada em várias plataformas distintas, dentre elas os dispositivos portáteis, onde foi feita a programação e as integrações desta pesquisa. Em resumo, qualquer dispositivo que tenha uma JVM instalada pode executar softwares criados em Java, esse fato permite que códigos possam ser reutilizados dinamicamente independente da plataforma, posteriormente a vida do desenvolvedor é facilitada.

## 2.4.4 Android

O Android é um sistema operacional que foi construído com base no sistema operacional Linux. Foi desenvolvido pela *OpenHandset Alliance*, uma aliança entre várias corporações de grande porte na área de tecnologia, dentre elas a *Google, Dell, Intel, Samsung, Nvidia*, entre outras. Por ser uma plataforma O*pen Source*, está disponível gratuitamente para que os fabricantes personalizem, não há configurações fixas de hardware e software (LEE, 2011). Devido a popularidade e fácil acesso dos dispositivos *Android*, foi definida a utilização dessa plataforma para o desenvolvimento da aplicação.

A arquitetura da plataforma é um montante de aplicações com base *Linux* de código aberto criado para diversos fatores e formas (ANDROID DEVELOPERS GUIDE, 2018). A divisão desse agrupamento consiste em *System Apps* na qual ficam os aplicativos nativos como Calendário, Câmera, Contatos, entre outros. *Java API Framework* que contém um conjunto completo de recursos do SO disponibilizados pelas APIs programadas na linguagem Java, esses blocos de código servem para facilitar o desenvolvimento das aplicações simplificando a reutilização de componentes e serviços já existentes. *Native C/C++ Libraries* contendo algumas bibliotecas nativas que são programadas emlinguagem C/C++, a maior parte dessas bibliotecas tem funcionalidades para manipulação de gráficos. *Android Runtime* que possui artefatos para gerirem as instâncias e os processos a serem executados visando baixo consumo de memória. A camada de abstração física (*Hardware Abstraction Layer*) faz o primeiro acesso ao componente físico implementando interfaces especificas do dispositivo, preparando-o para o uso pela Java API de maior nível. Por último a camada *Linux Kernel* que guarda módulos de segurança, gerenciamento de memória, energia, ou seja, a parte física em baixo nível.

****

**Figura 13** - Arquitetura da plataforma Android

**Fonte:** DEVELOPERS, 2018.

## 2.5 Ferramentas de Desenvolvimento

Para o desenvolvimento do sistema voltado para este projeto, será utilizado o Android Studio e para o versionamento do sistema, o GitHub.

Android Studio é uma IDE (*Integrated Development Environment*) de programação *OpenSource* desenvolvida pela JetBrains especificamente para desenvolvimento de aplicações móveis para Android. Ele foi apresentado à comunidade pela primeira vez em maio de 2013, com o intuito de substituir a IDE anterior nativa do Android, o Eclipse.

Atualmente a versão estável do Android Studio é a 3.1.2 que foi lançada em Abril deste ano (2018), e dá suporte à programação em Java, Python (mais utilizada para desenvolvimento relacionado com *Machine Learning*) e Kotlin (linguagem de programação criada e aprimorada pela própria Google com a promessa de ser menos verbosa e, portanto, mais ágil na hora de programar(CABRAL, 2018).

Neste projeto, será utilizado o Android Studio por ser nativo da Google e, também pela quantidade de tutoriais que existem na internet que podem auxiliar no desenvolvimento de aplicativos e uso das APIs. Além disso, trata-se de uma das IDEs mais robustas existentes atualmente (ANDROID DEVELOPERS GUIDE, 2018).

O GitHub é uma plataforma de armazenamento de código distribuído que utiliza Git, permitindo assim que o código seja baixado para ser trabalhado/alterado em computadores distintos e depois enviados para um servidor central, no qual é criado uma versão nova com as alterações feitas e que pode ser revisado pelos outros componentes do time, que podem fazer um *code-review* (revisão do código) para garantir a qualidade do código e que nada do que foi alterado “quebrará” a versão anterior.

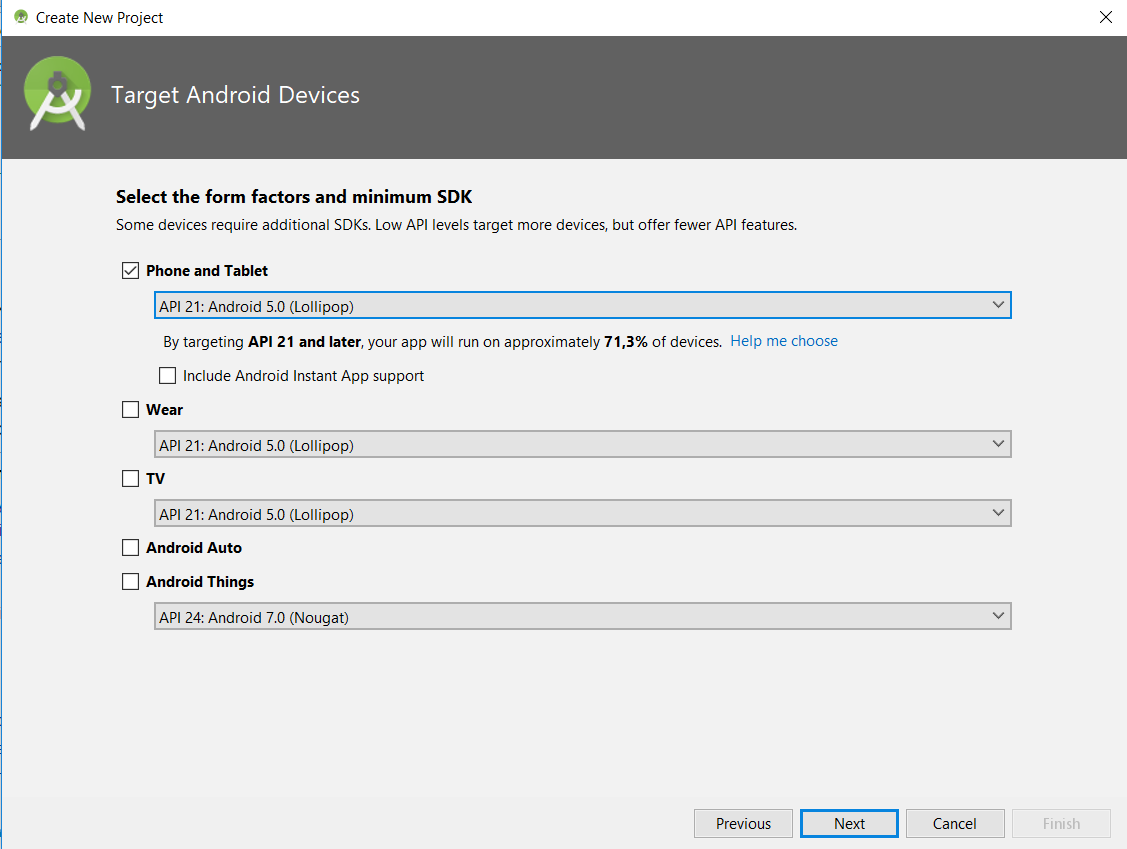
Ele mescla as funcionalidades de um SCM (*Source Control Management*) e de um DVC (*Distributed Version Control*). O SCM não é necessariamente um controle de versão de código, mas sim de qualquer arquivo que haja necessidade de versionar (como por exemplo um TCC, em que várias pessoas podem estar editando partes diferentes ao mesmo tempo e eventualmente precisar ver no que o outro está trabalhando). O próprio Wikipedia pode ser considerado um site com SCM, já que outras pessoas podem revisar o que está escrito e alterar se assim desejar. Já o DVC é um controle de versionamento onde você consegue ter um histórico completo das alterações em qualquer computador que esteja desenvolvendo o código. Isso te permite criar *branches* (seriam rotas alternativas para que não seja comitado (enviado) direto na *branch* principal) e fazer merges, aumentando a capacidade e a velocidade de desenvolvimento.

**2.5.1 Android Studio**

O Android Studio é muito útil, pois além de ser uma IDE de desenvolvimento, ainda possui alguns emuladores de smartphones nativos, ajudando desenvolvedores que não possuem capacidade de comprar vários smartphones para realizar testes a testar em smartphones virtuais.

Ao desenvolver no Android Studio, pode-se escolher qual linguagem de programação deseja-se utilizar para criar seu aplicativo além de quais versões do Android deseja suportar (quanto mais antigas, mais smartphones são suportados, porém algumas funcionalidades novas não serão possíveis de serem implementadas, já que elas só foram adicionadas em versões mais novas).

Neste presente projeto, busca-se um suporte para a maior quantidade possível de aparelhos, desde que suportem as APIs que serão utilizadas como Text-to-Speech e CloudVision. Observando os pré-requisitos para utilização das APIs e buscando o que foi implementado em cada versão (o que foi aprimorado nas APIs), percebe-se que seria necessário a API level 21 do Android pelo TTS (pelo fato do reconhecimento da linguagem da região) e da API level 19 a 20 que é pré-requisito mínimo de uso da CloudVision. Isso nos deixa com a versão mínima de uso do Lollipop (versão 5.0 do Android) com alcance de mais de 70% de usuários do Android (segundo o próprio Android Studio vide figura 14).

**Figura 14** - Android Studio

**Fonte:** ANDROID STUDIO, 2018.

**2.5.2 GitHub**

Para introduzir o GitHub é necessário saber alguns conceitos iniciais que seriam de *branch, commit, merge, pull, push* e *clone*.

Ao iniciar um trabalho, deve-se fazer um clone (clonar todas as informações do servidor na sua máquina). Após realizado este clone, toda vez que quiser verificar se seu código está atualizado, deve fazer um *pull* (puxa os arquivos do servidor web para sua máquina). Se quiser enviar suas alterações para o servidor, deve-se fazer um *push* e assim, enviar suas alterações.

Para não ter perigo de enviar algo direto para o servidor e “quebrar” o código, caso tenha ido com algum erro que você não percebeu anteriormente, é comum que se façam *branches*.

Uma *branch* recebe este nome por causa da relação com seu significado. Seria uma raiz principal (onde fica o código principal que vai ser alterado) da qual saem outras raízes (*branches* secundárias) que podem ser utilizadas para armazenar um código que um membro do time alterou (para enviar o código para esta *branch*, o usuário faz uma ação conhecida como *commit*). Após os outros membros do time fazerem um *code-review* (uma revisão no código garantindo a qualidade do software) é então realizado um *merge* (uma junção da *branch* secundária com a *branch* principal, fazendo com que agora a *branch* principal possua também as alterações feitas com as melhorias necessárias). Caso algo tenha sido feito errado, existe uma maneira de voltar atrás das alterações que é conhecido como *rollback*, e neste caso voltam todas as alterações até o último *commit*. (GIT-SCM, 2018)

## 2.6 API

Nos próximos tópicos, serão apresentadas algumas APIs que serão consumidas neste projeto, mas antes disso, será esclarecido o conceito de API.

API é uma sigla que significa *Application Programming Interface*, ou seja, Interface de programação para aplicativos (tradução livre). O conceito de API é bem amplo, mas segundo post no DevMedia (2018): “API é um conjunto de padrões de programação que permitem a construção de aplicativos e a sua utilização.”

Uma API possui classes, métodos e campos que deverão ser utilizados. Esta API pode ser de baixa ou alta complexidade, dependendo de quem a projetou, mas é dever do criador da API fazer algo que é intuitivo, fácil de ser utilizado por desenvolvedores (ou seja, pessoas que já tenham conhecimento técnico), e além disso, o mais genérico possível, já que ela deve interligar várias funções de um aplicativo, e geralmente por vários tipos de aplicativo.

Uma API é considerada boa quando ela é compreensível pelos desenvolvedores que implementam e consomem a API, se ela é consistente com aquilo que ela propõe, se os métodos são fáceis de usar, se ela possui boa documentação e, principalmente, se ela é autossustentável, ou seja, se os custos dela são equivalentes ao retorno que ela provê.

Neste projeto, foi escolhida a utilização de uma API em vez de criá-la, justamente pela complexidade de criação e facilidade do uso. Decidiu-se focar principalmente em trazer a melhor experiência possível para o usuário. Ainda segundo DevMedia (2018), a primeira API não fica boa, ela deve ser testada e estudada com relação à escalabilidade, e como estamos trabalhando com *Machine Learning*, teríamos que coletar uma massa de dados gigantesca, que não precisaremos se utilizarmos as da Google que já estão prontas e testadas. No caso da Google Cloud Text-to-Speech, ainda está na versão Beta, mas ao pesquisar percebemos que já está muito bem desenvolvida e documentada.

## 2.6.1 Google CloudVision

A Google Cloud Vision API (GCV) é uma ferramenta *Open Source* da Google que é bem prática de ser utilizada, pois já possui poderosos modelos de *Machine Learning* encapsulados.

Esta ferramenta visualiza uma imagem e buscando em um banco de dados criado previamente (você pode criar metadados ou usar o banco de dados da Google), reconhece o que está sendo apresentado para ela. No nosso caso, esta será empregada para fazer a conversão de imagem em texto, e após isso, com outra API da Google, conhecida como Google Speech, o texto será convertido em voz. Esta API será apresentada no tópico adiante.

Ela também possui integrado um recurso de reconhecimento óptico, também conhecido como OCR (*Optical Character Recognition*) que identifica os caracteres e o idioma automaticamente de textos escritos em imagens.

Apesar de ser uma API *Open Source*, não significa que ela seja de graça, principalmente se a quantidade de utilização dos recursos da ferramenta for muito grande. Mas em geral, para até 1000 acessos por mês, atualmente todos os recursos são gratuitos. Verificou-se em uma pesquisa que para este projeto, a detecção de textos custa US$1,50 a cada 1000 usos por mês para 1001 a cinco milhões de acessos e US$0,60 a cada 1000 usos por mês para cinco milhões e um até 20 milhões.

Conclui-se desta forma que para a utilização neste projeto, esta ferramenta não terá custos no início e caso haja algum eventualmente, pode-se colocar alguns anúncios ou estudar uma outra forma de remuneração para cobrir os custos de manutenção do *software*.

## 2.6.2 Google Cloud Text-to-Speech

Outra ferramenta da Google que será utilizada neste projeto, será a Google Cloud Text-to-Speech API (TTS), na qual é possível converter o texto em áudio com a vantagem de ter a sua implementação e utilização mais simples em relação a Google Cloud Vision API.

Estando disponível em diversos idiomas, inclusive o Português, esta ferramenta é capaz de converter o texto (obtido através da imagem pelo GCV) para áudio, que poderá ser ouvido através do smartphone do usuário. Apesar de ser simples de implementar, é uma ferramenta muito poderosa que se baseia em pesquisas de *WaveNet* e redes neurais (já abordado neste trabalho) para entregar com a melhor fidelidade possível (GOOGLE CLOUD, 2018).

É uma ferramenta em expansão, mas que já está disponível em mais de 30 vozes e 12 idiomas, possível de alterar a velocidade de leitura para até quatro vezes mais rápido, sendo possível aumentar o som de saída em até 16 decibéis.

O valor desta ferramenta é também considerado baixo, assim como da GCV, sendo gratuito para até um milhão de caracteres mensais lidos (para vozes *WaveNet*), ou quatro milhões de caracteres mensais (para vozes que não são *WaveNet*), e sendo cobrado US$4,00 e US$16,00 para cada um milhão de caracteres que ultrapassarem o uso.

Concluímos que pela facilidade de uso, e pelo poder da ferramenta, esta é uma API indispensável para o nosso projeto, pois já foi desenvolvida e testada por muitas empresas grandes (como a própria Google, Rovio, Mistui, Evernote, Khan Academy, entre muitas outras) e continua em expansão com potencial de crescimento enorme, além de ser nativo Android (que é o nosso foco atual).

## 2.6.3 Google App Engine

Repositórios de dados são conhecidos como coleções de informação digital, os mesmos podem ser desenvolvidos de diferentes maneiras e com propósitos distintos. Podem-se caracterizar por ser colaborativos, com um controle não tão intenso das informações e da autoridade dos documentos, tal como os repositórios dirigidos ao público em geral. Mas podem, também, ter um alto nível de controle em sua arquitetura, condensando o acesso as informações, exigindo uma complexidade maior para o acesso aos dados, que é o caso da Google app engine.

Esse mecanismo criado pela empresa Google, tem como principal objetivo fazer o controle das requisições das APIs que consomem as informações armazenas para o funcionamento correto das funcionalidades propostas, isso se aplica aos mecanismos citados anteriormente (GCV e TTS). Segundo Google 2018:

Os SDKs do Firebase para Cloud Storage usam o repositório padrão do Google App Engine. Isso significa que, quando um app do App Engine é criado, as App Engine APIs incorporadas podem ser usadas para compartilhar dados entre o Firebase e o App Engine. Isso é útil para executar codificação de áudio, transcodificação de vídeo e transformações de imagem, bem como outros processamentos de computação intensiva de segundo plano.

# Análise e Desenvolvimento

Neste capítulo será apresentado o desenvolvimento das ideias para um aplicativo a partir das técnicas descritas no capítulo anterior. Busca-se com ele atender as necessidades de um público alvo carente de algumas tecnologias possibilitando “novas visões”.

## 4.1 Público alvo

Nosso trabalho visa atingir o nicho de pessoas que possuem dificuldades ou deficiências visuais. Estas pessoas podem possuir cegueira, baixa visão, visão subnormal, entre outras.

Além deste público alvo, nosso projeto pode também atender pessoas não alfabetizadas, possibilitando elas a ouvir o conteúdo escrito.

.

## 4.2 Ouvindo o público alvo

A etapa “Ouvir” iniciou-se através de conversas com pessoas com deficiência visual onde os problemas de sua rotina foram apresentados. As questões realizadas e algumas respostas transcritas estão disponíveis no Apêndice B. Desta forma, constatou-se alguns problemas como:

* Dificuldades em ler o cardápio
* Identificar imagens
* Falta de ferramentas voltadas para o público alvo
* Usabilidade de smartphones
* Falta de aplicativos
* Entre outros;

## 4.3 Ideação

Com os problemas do público alvo identificados, iniciou-se a fase de ideação. Foram realizadas algumas reuniões para se discutir possíveis “soluções” e chegou-se a definição do desenvolvimento de um aplicativo mobile o qual faria a conversão de uma imagem em áudio, e, para isto, seria necessário primeiramente converter a imagem em texto e, depois, o texto em áudio.

Com estas informações, foi possível elaborar as fichas com os requisitos funcionais e não funcionais disponíveis no apêndice A, além de diagramas de caso de uso, atividade e classe que estão disponíveis no apêndice C.

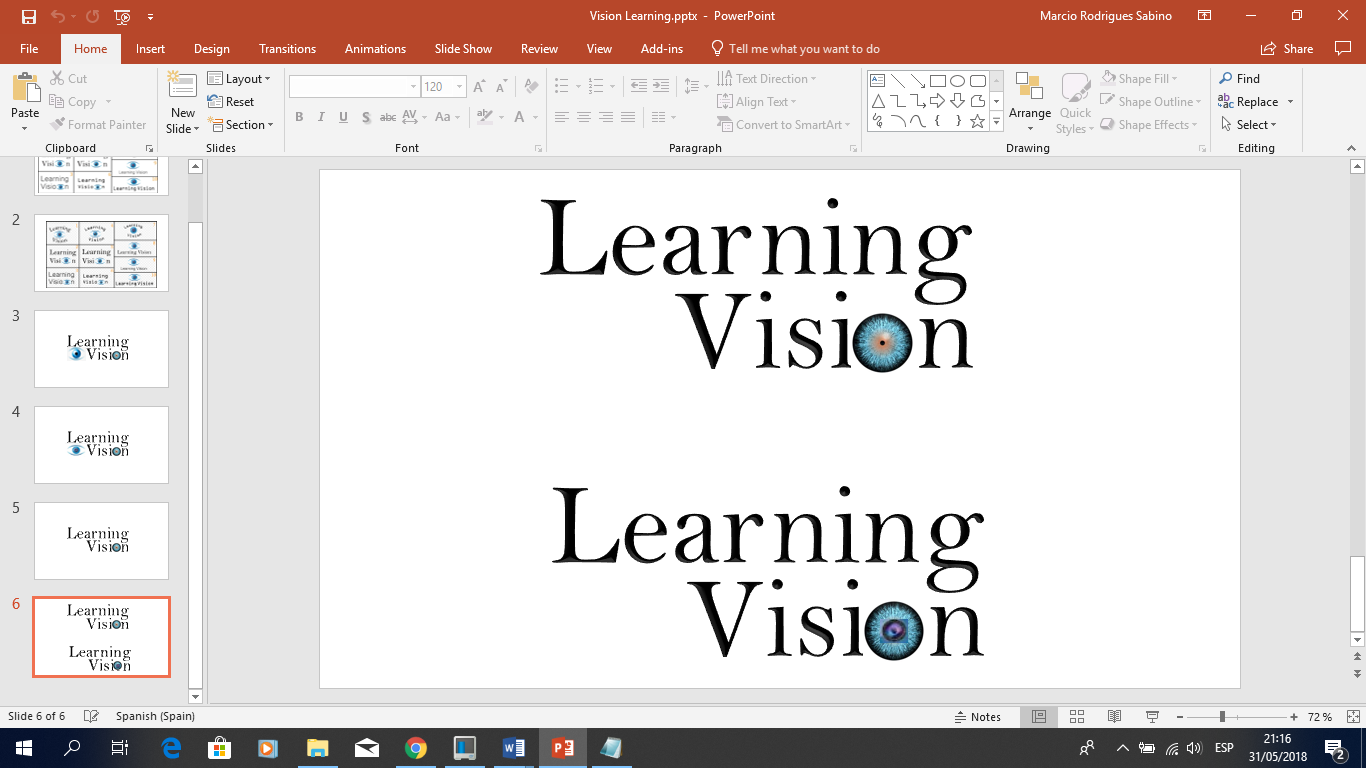
Ao serem realizados os estudos, ponderou-se a viabilidade da implementação de ferramentas conhecidas com *Wearables,* que incluem dispositivos inteligentes como objetos vestíveis para o ser humano. Essa tecnologia é aplicada a relógios, camisetas, luvas, óculos entre outros. Considerando a aplicabilidade do projeto a mecanismos direcionados a visão, os óculos (*SmartGlass*) foram selecionados como uma possibilidade diferencial levando em conta a integração do mesmo com o *software* e a ascensão deste tipo de tecnologia no mercado. Como artefato final, após uma posterior análise de viabilidade de valores, conectividade e diferenciabilidade será feita a implantação de acordo com a facticidade da mesma.

Para Labbate (2001) :

As possibilidades para os *wearables* no mercado são ilimitadas e é animador observar para onde empresas e designers estão levando essa tecnologia. Assim como a previsão dos analistas do setor, acreditamos que a tecnologia *wearable* trará ganhos exponenciais em satisfação e produtividade tanto para os representantes de serviço ao cliente como para os funcionários da força de trabalho em campo.

Depois de definidas possibilidades viáveis de solução quanto à parte financeira, tempo para o desenvolvimento e viável tecnologicamente, iniciou-se as discussões a respeito de um possível nome para o aplicativo decidindo-se pelo nome Learning Vision fazendo uma alusão à *machine learning* voltada à visão computacional. Além disso, foi debatida uma identidade visual para o aplicativo de forma que este quando observado trouxesse indicativos da função do app. A figura 15 apresenta o logo representativo do aplicativo a ser desenvolvido neste projeto:

**Figura 15** - Logo



**Fonte**: Próprio Autor

Observe que a letra “O” foi modificada com o formato de uma íris com uma câmera no seu centro fazendo uma alusão a “virtualização” da visão.

## 4.4 Gerenciamento do projeto

O projeto foi particionado em duas grandes fases a serem executadas ao longo do ano de 2018. Através da metodologia *design thinking* estipulou-se etapas e objetivos a serem seguidos. Estes eram propostos ou conversados com o Orientador em intervalos semanais ou quinzenais, tornando mais eficaz a viabilidade tempo do projeto.

As partes de desenvolvimento e criação de telas serão iniciadas logo após a qualificação deste projeto. Nestas, serão utilizadas as ferramentas e as APIs estudadas e viabilizadas até o momento.

A tabela 4 apresenta o cronograma das atividades planejadas para o desenvolvimento deste projeto.

**Tabela 4** - Cronograma

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ATIVIDADE** | **Fev** | **Mar** | **Abr** | **Mai** | **Jun** | **Jul** | **Ago** | **Set** | **Out** | **Nov** | **Dez** |
| Definição do grupo | x |  |  |  |  |  |  |  |  |  |  |
| Definição do tema | x | x |  |  |  |  |  |  |  |  |  |
| Introdução |  | x |  |  |  |  |  |  |  |  |  |
| Referencial Teórico |  | x | x | x | x |  | x |  | x |  |  |
| Entrevista com Público Alvo |  | x |  |  |  |  |  | x |  |  |  |
| Fichas de requisito |  | x | x |  |  |  |  |  |  |  |  |
| Diagramas |  |  | x | x |  |  |  |  |  |  |  |
| *Brainstorming* |  | x | x |  |  |  | x |  |  |  |  |
| Apresentação - qualificação |  |  |  |  |  | x |  |  |  |  |  |
| Protótipos de telas |  |  |  |  |  | x | x | x |  |  |  |
| Programação |  |  |  |  |  | x | x | x | x |  |  |
| Teste com o *software* |  |  |  |  |  |  |  |  |  | x |  |
| Conclusão |  |  |  | x |  |  |  |  |  | x |  |
| Apresentação - Defesa |  |  |  |  |  |  |  |  |  |  | x |

**Fonte**: Próprio Autor.

# conclusão

Atualmente já existem alguns aplicativos que são desenvolvidos pensando na acessibilidade de pessoas com deficiência visual, porém ainda são muito poucos. O nosso público alvo, não visa auxiliar no uso do smartphone e sim auxiliar no dia-a-dia do usuário, possibilitando a leitura de cardápios, de placas com textos, de embalagens, entre outros objetos, diminuindo a dependência de auxílio dessas pessoas, dando mais liberdade a elas.

Até o presente momento, após um desejo inicial de tema de pesquisa, utilizou-se da metodologia *design thinking* para dar vida à um projeto bem fundamentado e viável frente aos quesitos tempo, financeiro e exequibilidade. Foram coletadas e analisadas informações sobre as necessidades de pessoas com deficiência visual, identificando-se necessidades que poderiam ser tangenciadas para uma melhora com um auxílio à visão. A proposta foi modelar um aplicativo mobile que auxilie a leitura e interpretação do mundo, tentando minimizar os obstáculos encontrados por quem tem algum tipo de deficiência (de visão, ou até mesmo de leitura).

A continuidade deste projeto se dará ao longo do segundo semestre de 2018 no qual serão desenvolvidos: protótipos de telas do app, programação do *software* eteste para validação do aplicativo.

Espera-se ao final do projeto realizar um teste com uma amostra do público alvo e ter o retorno de algo útil às suas vidas.

Referências Bibliográficas

ACHARYA, T., RAY, A. K. **Image Processing- Principles and Applications**. John

Wiley & Sons, Inc. 2005.

ADDICAM, S. MALIK, S. and TIAN, P**. Building Itelligent Systems**. Intel Press., 2012.

ALBUQUERQUE, Márcio Portes de; ALBUQUERQUE, Marcelo Portes de. **Processamento de Imagens: Métodos e Análises**. Rio de Janeiro: Centro Brasileiro de Pesquisas Físicas – Cbpf/mct, 2004.

ANDRADE, Felipe. **INFO ESCOLA**. Disponível em: http://www.infoescola.com/informatica/o-que-sao-linguagens-de-programacao Acesso em: 30 de Maio de 2018.

**ANDROID STUDIO RELEASE NOTES**. Disponível em: <https://developer.android.com/studio/releases/> Acesso em: 25 de maio de 2018.

**ANDROID DEVELOPER GUIDE.** Disponível em: https://developer.android.com/guide/platform/. Acesso em: 30 de abril de 2018.

**APPLICATION PROGRAMMING INTERFACE: DESENVOLVENDO APIS DE SOFTWARE**. Disponível em: <https://www.devmedia.com.br/application-programming-interface-desenvolvendo-apis-de-software/30548> Acesso em 20 de maio de 2018.

BATISTA, Leonardo Vidal. **Introdução ao Processamento Digital de Imagens**. João Pessoa, Pb: Notas de Aula, 2005. 84 slides, color.

BECK, Kent et al. **Manifesto para o desenvolvimento ágil de software**. 2001. Disponível em: <http://www.manifestoagil.com.br/index.html>. Acesso em: 16 maio 2018.

BOLDRINI, J. L. **Álgebra Linear**. 3. ed. São Paulo: Editora HARBRA Ltda., 1986.

CABRAL, Carlos. **Kotlin:**Um jeito novo de escrever aplicativos multiplataforma. 2018. Disponível em: <https://abacomm.com.br>. Acesso em: 14 maio 2018.

CARLO, James J. Di Carlo. ZOCCOLAN, Davide. RUST, Nicole C. **How Does the Brain Solve Visual Object Recognition?** Neuron. Volume 73, Issue 3, p.415-434, 9 de fevereiro de 2012.

CLARO, Daniela; SOBRAL, João Bosco. **Programação em Java.** 1ª edição, Florianópolis: Editora Pearson, 2008.

**CLOUD-TO-SPEECH**. Disponível em: <https://cloud.google.com/text-to-speech/> Acesso em 20 de Maio de 2018.

**CLOUD VISION API**. Disponível em: <https://cloud.google.com/vision/?hl=pt-br> Acesso Acesso em 20 de Maio de 2018.

COPELAND, Michael. **What’s the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?** 2016. Disponível em: <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>. Acesso em: 4 de Abril de 2018.

DEVELOPERS. **Arquitectura de la plataforma.** Disponível em: <Arquitectura https://developer.android.com/guide/platform/de la plataforma>. Acesso em: 25 abr. 2018.

**Documentation of Text-to-Speech**. Disponível em: <https://developer.android.com/reference/android/speech/tts/TextToSpeech> Acesso Acesso em 20 de Maio de 2018.

FORSYTH, D., PONCE, J. **Computer Vision: A modern approach**. Prentice Hall, 2001.

**Git Distribuído: Contribuindo Para Um Projeto.** Disponível em: <https://git-scm.com/book/pt-br/v1/Git-Distribu%C3%ADdo-Contribuindo-Para-Um-Projeto> Acesso em 30 de Maio de 2018

GONZALEZ, R. C. WOODS, R. E. **Digital Image Processing.** 3rd Edition. Prentice-Hall, Inc., 2006.

GOUSSEVSKAIA, Olga N. et al. **Internet das Coisas: da Teoria à Prática**. UFMG, 2017.

GRUMAN, Galen. **IoT é um grande e confuso campo à espera de explodir**. Computer World, 2014.

HARNAD, Stevan. **The Annotation Game:**On Turing (1950) on Computing, Machinery, and Intelligence. 2008. Disponível em: <https://eprints.soton.ac.uk/262954/1/turing.html>. Acesso em: 15 de abril de 2018.

**Integrar com o Google Cloud Platform.** Disponível em:< https://firebase.google.com/docs/storage/gcp-integration?hl=pt-br> Acesso em 9 de junho de 2018

JACQUES JUNIOR, Julio C. S.. **Processamento de Imagens: fundamentos**. Porto Alegre, Rs: Notas de Aula, 2014. 38 slides, color.

JÄHNE, B. **Digital Image Processing**. Springer-Verlag, 2002.

JOLY, Martine. **Introdução à Analise Da Imagem**. Papirus Editora, 11 edição, 2007.

LABBATE , Alexsandro. **Tecnologia wearable: moda, roupas inteligentes e o futuro dos vestíveis.** 2016. Disponível em: https://canaltech.com.br/mercado/tecnologia-wearable-moda-roupas-inteligentes-e-o-futuro-dos-vestiveis-58795/. Acesso em: 30 de Maio de 2018.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de informação gerenciais**. 3ª edição. Rio de Janeiro: Editora LTC, 2001.

LEE, Wei-Meng. **Introdução ao Desenvolvimento de Aplicativos para Android.** 1ª edição Rio de Janeiro: Editora Ciência Moderna Ltda, 2011.

LIMA, E. L**. Espaços Métricos**. 3. ed. Rio de Janeiro: Projeto Euclides, 1993.

LOUREIRO, A. A., et Al. **Redes de Sensores Sem Fio**. Simpósio Brasileiro de Redes de Computadores (SBRC), p. 179–226, 2003.

LOPES, J. G. **Introdução ao Machine Learning e seus principais algoritmos**. 2018. Disponível em: <http://joseguilhermelopes.com.br/introducao-ao-machine-learning-e-seus-principais-algoritmos/>. Acesso em: 05 maio 2018.

LUCENA, Oeslle A. S.; VELOSO, Luciana R.; LOPES, Waslon T. A.. **Reconhecimento de Objetos em Imagens. Encontro Anual do IECOM em Comunicações, Redes e Criptografia: ENCOM**. Fortaleza, p. 1-2. out. 2016.

**MACHINE Learning: entenda a aplicação em diferentes mercados Udacity Brasil. entenda a aplicação em diferentes mercados Udacity Brasil**. 2017. Disponível em: <https://br.udacity.com/blog/post/aplicacoes-machine-learning>. Acesso em: 4 de abril de 2018.

MARQUES, Jair Mendes; NETO, Anselmo Chaves. **Discriminação, Classificação e Reconhecimento de Padrões**. Curitiba, PR, Notas de Aula, 2013.

MARR, Bernard. **What Is The Difference Between Artificial Intelligence And Machine Learning?**2016. Disponível em: <https://www.forbes.com/sites/bernardmarr/2016/12/06/what-is-the-difference-between-artificial-intelligence-and-machine-learning/#6bd0ade82742>. Acesso em: 4 de Abril de 2018.

MCCLELLAND, Calum. **The Difference Between Artificial Intelligence, Machine Learning, and Deep Learning.**2017. Disponível em: <https://medium.com/iotforall/the-difference-between-artificial-intelligence-machine-learning-and-deep-learning-3aa67bff5991>. Acesso em: 4 de abril de 2018.

MITCHELL, Tom M. **Machine Learning.**New York: Mcgraw-hill Science/engineering/math, 1997.

MONTEIRO, João Bosco. **Google Android crie aplicações para celulares e tablets**. 1ªedição, São Paulo: Editora Casa do Código Ldta, 2015.

MOREIRA, Nelma. **Processamento de Imagem**. 2000. Disponível em: <https://www.dcc.fc.up.pt/~nam/aulas/0001/pi/trabalho2/trab2/trab2.html#SECTION00012000000000000000>. Acesso em: 01 jun. 2016.

News Medical Life Sciences. **Como o cérebro reconhece objetos**. Disponível em: https://www.news-medical.net/news/2006/01/18/40/Portuguese.aspx. Acesso em: 25 de Março de 2018.

NÖTH, Winfried; SANTAELLA, Lucia. **Imagem: cognicão, semiótica, mídia**. Editora Iluminuras, 2015.

PEREIRA, Vinicius de Novaes; PESCO, Dirce Uesu; BORTOLOSSI, Humberto José. **Matrizes e Imagens Binárias: Matemática: números e operações**. Disponível em: <http://www.uff.br/cdme/matrix/matrix-html/matrix\_boolean/matrix\_boolean\_br.html>. Acesso em: 10 maio 2015.

PRESS, Gil. **Internet of Things By The Numbers: Market Estimates And Forecasts**. FORBES, 2014.

PRESSMAN, Roger S. **Engenharia de Software:**uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de software**: uma abordagem profissional. 8 ed. Porto Alegre: AMGH, 2016.

QUEIROZ, José Eustáquio Rangel de; GOMES, Herman Martins. **Introdução ao Processamento Digital de Imagens**. Revista de Informática Teórica e Aplicada - (RITA), Volume VIII, Número 1, Rio Grande do Sul, 2001.

RAMOS, Ronaldo F. **Mini Curso de Reconhecimento de Padrões / Data Mining / Aprendizagem de Máquina. Aula número 1**. Fortaleza, Ce, 2014. Color. Disponível em: <https://www.youtube.com/watch?v=5DDlv06dpxY&list=PLlvmgEJXhL3yKLi4EOHvPF1Uv4g4Piuro&index=2>. Acesso em: 02 jun. 2016.

ROSEN, Jill. **How the brain instantly separates trash from treasure. Science+Technology.** Diponível em: https://hub.jhu.edu/2018/02/08/brain-identifies-valuable-items/. Acesso em: 26 de Março de 2018.

RUSS, J. C. **The image processing handbook**. CRC Press LLC, 2000 3rd ed.

SAMUEL, A. L.. **Some Studies in Machine Learning Using the Game of Checkers**. Ibm Journal Of Research And Development, [s.l.], v. 3, n. 3, p.210-229, jul. 1959. IBM. http://dx.doi.org/10.1147/rd.33.0210.

SANTOS, B.P. et Al. **Internet das Coisas: da teoría à Prática**. Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuidos (SBRC), 2016.

SBROCCO, José Henrique Teixeira de Carvalho; MACEDO, Paulo Cesar de. **Metodologias ágeis:**engenharia de software sob medida. São Paulo: Érica, 2012.

SEBRAE. **Entenda o design thinking.**

Disponível em: <http://www.sebrae.com.br/sites/PortalSebrae/artigos/entenda-o-design-thinking,369d9cb730905410VgnVCM1000003b74010aRCRD>. Acesso em: 03 out. 2017.

SIMONITE, Tom. **Supercomputador do Baidu reconhece imagens melhor que o ser humano.** Disponível em: <https://www.technologyreview.com/s/537436/baidus-artificial-intelligence-supercomputer-beats-google-at-image-recognition/> Acesso em: 28 de Março de 2018.

SOMMERVILLE, Ian. **Engenharia de Software.**9. ed. São Paulo: Pearson Prentice Hall, 2011.

**TIOBE.** Disponível em: <https://www.tiobe.com/tiobe-index/>. Acessado em: 30 de abril de 2018.

TUCKER, Allen; NOONAN, Robert. **Linguagens de Programação Princípios e Paradigmas.** 2ª edição, São Paulo: Editora AMGH Ltda, 2010.

TURING, Alan M. **Computing Machinery and Intelligence**. **Mind**, [s. L.], v. 59, n. 236, p.433-460, out. 1950.

VIANNA, Maurício et al. **Design thinking: inovação em negócios**. Rio de Janeiro: Mjv Press, 2012.

APÊNDICE A – Documento de requisitos

Disponível no CD anexo ao trabalho.

Repositório com os dados:

https://github.com/LearningVisionFatec/Documents



APÊNDICE B – Questões para entrevista com pessoas com deficiência visual

Abaixo, apresenta-se algumas questões direcionadas à pessoas com deficiência visual na fase de imersão do *design thinking*:

* Qual tipo de deficiência visual você possui?
* Cegueira Total – Cegueira Parcial – Deficiência Visual como miopia, astigmatismo, etc.
* De 0 a 10, sendo 0 pouco adaptada e 10 muito adaptada, marque o quanto você acha que a tecnologia é adaptada para pessoas com algum tipo de deficiência visual.
* Você possui um smartphone?
  + Sim
  + Não
* De 0 a 10, sendo 0 muito difícil e 10 muito fácil, marque o seu nível de dificuldade com relação ao uso do smartphone.
* Quais suas maiores dificuldades com relação ao uso do smartphone?
* De 0 a 10, sendo 0 nada relevante, e 10 muito relevante, marque o quanto você acha que um aplicativo que faça a conversão de imagem para áudio seria relevante no seu dia a dia.
* Caso se sinta à vontade, comente sobre suas dificuldades e o que acredita que a tecnologia poderia lhe trazer como beneficios.

APÊNDICE C – Diagramas

Disponível no CD anexo ao trabalho.

ANEXO D – Respostas de um dos entrevistados

A seguir veremos as respostas de um entrevistado. O mesmo enviou um áudio, e utilizamos um aplicativo de transcrever o áudio em texto para inserir no trabalho. Algumas correções foram efetuadas para melhorar o entendimento do texto:

“Beleza! vamos lá em relação ao uso de smartphone.

Sim, hoje em dia tem smartphones mais modernos, tem alguns que já vem com recursos de acessibilidade.

Antigamente, era necessário baixar um aplicativo que faria a leitura para você, mas hoje em dia a maioria já vem com este aplicativo. No iPhone, por exemplo, que, pelos cegos conheço e ouço falar, é o que é mais acessível, e que mais facilita a gente, mais simples e a gente consegue ter uma Independência bem, bem significativa com ele. Eu mesmo uso WhatsApp, Facebook, etc.

Em relação a uma tecnologia que codificasse imagens e transcrevesse em texto de alguma forma, seria excepcional para os cegos. É onde estão começando a chegar.

Por exemplo, se você me manda uma imagem ou uma foto no Smartphone, eu sei que é uma foto, mas eu não sei o que tem na foto, entendeu?

No Facebook já está acontecendo de dar algumas prévias sobre o que é a imagem, entretanto, eu não sei se isso é uma coisa do Facebook, ou se é uma coisa do voiceover, que é o recurso de acessibilidade para cegos que vem no iPhone. Mas algumas fotos que eu recebo, o meu telefone fala uma coisa como: “paisagem”, “área externa”, “área interna”, “noite”, “desfocada”, “multidão”, etc.

Uma tecnologia que, através de uns óculos ou de alguma coisa, mirasse em figuras e transcrevesse isso de alguma forma, seria aquela coisa mais imediata. Para cardápios em restaurante, por exemplo, seria algo excepcional.”

1. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [↑](#footnote-ref-2)