UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

"Projeto de *software* com varredura automática e controle de dispositivo robótico"

Vanderlei de Oliveira Júnior

Orientadora: Prof.ª. Dr. ª Erica Regina Daruichi Machado

Trabalho de Graduação apresentado à Faculdade de Engenharia - UNESP – Campus de Ilha Solteira, para cumprimento de requisito para obtenção do Grau de Engenheiro Eletricista.

Ilha Solteira – SP

12/2020

AGRADECIMENTOS

Agradeço primeiramente à Deus e à minha família, em especial meus pais e minha irmã por ter me dado todo o suporte nesta empreitada, tanto emocionalmente quanto financeiramente enquanto residia em Ilha Solteira - SP.

Em segundo lugar agradeço à professora dra. Erica R. D. Machado pela orientação de tema, revisão e possibilidade de realizar o meu trabalho remotamente e ao professor dr. Marcelo Sanches pelo auxílio e orientação sobre o estágio que realizei. Agradeço também aos demais professores da FEIS pelos quais eu tive a honra de aprender, assim como demais funcionários como técnicos envolvidos nos laboratórios e trabalhadores do setor de graduação.

Agradeço também em especial à minha companheira Laura Manuci pelo suporte emocional e psicológico, principalmente após eu terminar as disciplinas presenciais. Por fim, mas não menos importante, agradeço aos meus amigos que fiz na faculdade, em especial meu caro Luiz Henrique, a parceria que tivemos, tanto de *hobbies* quanto de estudos, tornou a experiência em Ilha Solteira um período bem marcante na minha vida.

SUMÁRIO

1 INTRODUÇÃO 10
2 OBJETIVOS 12
3 REVISÃO BIBLIOGRÁFICA 13
3.1 Educação Inclusiva de Pessoas com Deficiências 13
3.1.1 Recursos Tecnológicos a Serviço da Inclusão de PCD 13
3.1.2 Varredura Automática 14
3.2 Engenharia de software 15
3.2.1 Qualidade de *Software* 15
3.2.2 Processos de *Software* 16
3.2.3 Processo Unificado 26
3.2.4 Engenharia de Requisitos 27
3.2.5 Padrão UML 28
3.2.6 Modelagem 31
3.2.7 Projeto e Construção 36
3.3 Linguagem *Python* 36
3.3.1 Interface Gráfica do Usuário com *Tkinter* 37
3.3.2 Comunicação com *Hardwares* 38
3.4 Comunicação serial 38
3.5 Comunicação *wireless* 40
4 MATERIAIS E MÉTODOS 41
4.1 Estudos Teóricos 41
4.2 Análise de requisitos e modelagem 41
4.2.1 Requisitos de Acessibilidade 41
4.3 Implementação computacional 42
4.4 Design de Interfaces 42

4.5 Construção do protótipo e Testes 42
5 DISCUSSÃO E RESULTADOS 43
5.1 Modelagem de requisitos 43
5.1.1 Diagrama de Casos de Uso e Diagramas de Classes 43
5.1.2 Diagrama de Atividades 44
5.2 Implementação Computacional 45
5.2.1 Criação da Varredura Automática 46
5.2.2 Interface do menu principal 49
5.2.3 Interface do botão SOBRE 50
5.3 Interface do menu de controle e comunicação 51
5.4 Configuração do *Hardware* 52
6 CONCLUSÕES 61
6.1 Trabalhos Futuros 62
7 REFERÊNCIAS 63
8 ANEXOS 65
8.1 ANEXO A: Codificação do *software* considerando comunicação serial 65
8.2 ANEXO B: Código em arduino para compilação no hardware 79
8.3 ANEXO C: Mudanças no código em *python* para comunicação *wireless* 81

LISTA DE FIGURAS

Figura 1: Metodologia de processo de software. 18
Figura 2: Etapas de modelo em cascata. 19
Figura 3: Modelo em "V". 20
Figura 4: Modelo de desenvolvimento incremental 20
Figura 5: Modelo evolucionário de prototipação. 22
Figura 6: Modelo de espiral 23
Figura 7: Modelo de espiral simplificado 24
Figura 8: Elemento genérico de um modelo de processo concorrente 25
Figura 9: Processo unificado 27
Figura 10: Leitores de diferentes tipos de requisitos 28
Figura 11: Relacionamento de dependência 30
Figura 12: Associação plana 30
Figura 13: Associação de agregação simples 30
Figura 14: Associação de agregação por composição 31
Figura 15: Relacionamento de generalização 31
Figura 16: Exemplo de diagrama de caso de uso 32
Figura 17: Exemplo de diagrama de atividades 33
Figura 18: Exemplo de diagrama de sequência 35
Figura 19: Protocolo de transmissão serial assíncrona 39
Figura 20: Pontos de amostragem dos bits para recepção 39
Figura 21: Diagrama de casos de uso 43
Figura 22: Diagrama de classes 44
Figura 23: Diagrama de atividades 44
Figura 24: Projeto de interface do menu inicial 46
Figura 25: Código com exemplo de aplicação do método after() 47
Figura 26: Código da função temp1( ) 48
Figura 27: Interface do menu de varredura automática 49
Figura 28: Métodos transient( ) e grab_set( ) 49
Figura 29: Interface do menu principal 50
Figura 30: Tela da opção "SOBRE" 51
Figura 31: Interface do menu de controle do carrinho 51
Figura 32: Classe Application3 52

Figura 33: Função Motorup() 53
Figura 34: Função cima( ) 53
Figura 35: Circuito de ponte H 54
Figura 36: Pinagem da placa Arduino Uno R3 55
Figura 37: Pinagem de um microcomputador Raspberry Pi 1 B+ ou mais recente 56
Figura 38: Kit robótico montado 57
Figura 39: Nova configuração da função motorup( ) 57
Figura 40: Função init( ) 58
Figura 41: Projeto do circuito digital 59
Figura 42: Interface do conversor de Python para arquivo executável 60

LISTA DE SIGLAS

API ····· Interface de Programação para Aplicações
ASCII ····· Código Padrão Americano para Intercâmbio de Informações
DC ····· Corrente Contínua
GPIO ····· Pinos de Entrada e Saída de Uso Geral
GUI····· Interface Gráfica do Usuário
IDE ····· Ambiente de Desenvolvimento Integrado
PCD ····· Pessoa com Deficiência
PU ····· Processo Unificado
SSH ····· *Secure Shell* (protocolo de rede)
TA ····· Tecnologia Assistiva
UML ····· Linguagem de Modelagem Unificada
WLAN ····· Rede de Área Local *Wireless*
DFD ····· Diagrama de fluxo de dados

RESUMO

Os conceitos pertencentes à engenharia de *software* são fundamentais para profissionais que atuam com desenvolvimento de *software*s ou áreas correlatas. Com sua metodologia de projeto visando desde a busca dos requisitos até a implementação e revisão do produto final é possível apreender uma experiência útil para atuação nesta área, com o profissional podendo ampliar seu uso em maiores escalas. Realizou-se neste trabalho um projeto de *software* com finalidade de uso como ferramenta lúdica, didática e/ou de reabilitação por crianças com deficiências, com assistência de técnicas de Tecnologia Assistiva. Escolheu-se a linguagem *Python* para a sua elaboração, dada a sua praticidade e robustez oferecidas ao programador. Já para a interface gráfica o *Tkinter* foi a biblioteca escolhida por sua simplicidade e popularidade para projetos de pequeno porte, além de estar presente como biblioteca padrão da linguagem. A ideia central do projeto é apresentar um programa que possibilite ao usuário o controle de um pequeno robô automóvel. Inicialmente se pensou em utilizar uma comunicação serial atrelada a uma placa de microcontrolador *Arduino*, apresentando-se também uma proposta que possibilitasse comunicação *wireless* com a utilização de um minicomputador *Raspberry Pi* para com maior facilidade de interação com *Python* e melhor possibilidade de estabelecimento de controle remoto, mostrando-se no decorrer do trabalho como seria seu projeto para sua construção. Por fim, demonstra-se também as diferenças de projeto entre o uso dos dois hardwares com sugestões de melhorias para próximos trabalhos e instruções para implementá-los.

Palavras-chave: Tecnologia Assistiva, Engenharia de *Software*, interface gráfica do usuário, comunicação *wireless*, comunicação serial.

ABSTRACT

The concepts related to *software* engineering are fundamental to professionals who work with *software* development or other related areas. Thanks to project methodology that pursues from requisite to implementation and final product's revision is possible to apprehend a useful experience for work in this area, with the possibility to expand your application into larger scale. This work consists in a *software* project for use by disabled children as a ludic, didatic and rehabilitation tool with assistance from Assistive Technology techniques. The python language was chosen for its construction because of its practicality and effectiveness offered to the programmer. For graphic interface the Tkinter library was adopted for the sake of its simplicity and high popularity among small scale developers, besides being as a standard library of python. The project's central idea is presenting a program that allows the user to control a small car from distance. Initially it was thought with serial communication in addiction to the single-board microcontroller *Arduino*, proposing also the use of wireless communication through a mini computer such as Raspberry Pi in order to find greater interaction with python and better possibility of establishment of a remote control, also presenting how it should work. Finally, it is shown the differences between the two options, suggestions for next works and instructions for its implementation.

Key words: Assistive Technology, software engineering, graphic user interface, wireless communication, serial communication.

1 INTRODUÇÃO

Nas últimas décadas o conceito de inclusão social difundiu-se como elemento de minimização de desigualdades. Com a democratização de muitos países subdesenvolvidos no final do século XX esta passou a ser buscada através de políticas como ações, leis, decretos, entre outras medidas, visando a integração de pessoas excluídas da sociedade (BRASIL, 2018). Especificamente no Brasil, esta definição parte do direito à igualdade dado em sua Constituição.

De acordo com o artigo 2º do Estatuto da Pessoa com Deficiência (Lei 13.146/15), considera-se pessoa com deficiência (PCD) aquele indivíduo que, por razão de um impedimento físico, mental, intelectual ou sensorial, não possui participação em patamar de igualdade com outras pessoas (BRASIL, 2015).

Com estas definições iniciou-se a aplicação medidas de acessibilidade, para uso de serviços, produtos e informações, como elevadores para usuários de cadeiras de rodas, banheiros adaptados e rampas de acesso. A partir dos anos 80, com o desenvolvimento tecnológico, buscou-se maneiras de utilizar a tecnologia a serviço da acessibilidade, criando-se assim o conceito de Tecnologia Assistiva.

O termo "Tecnologia Assistiva" é utilizado para caracterizar dispositivos, instrumentos ou equipamentos que auxiliem na funcionalidade de pessoas com deficiência, reduzindo suas dificuldades para realização de determinadas atividades. Exemplos de Tecnologias Assistivas vão desde hardwares adaptados até funcionalidades de *software*, como aplicativos e atalhos de acessibilidade para acesso ás funções específicas para este fim. O termo é bem definido no Brasil segundo o Comitê de Ajudas Técnicas (CAT), órgão especializado nesta área pertencente à Secretaria Especial de Direitos Humanos da Presidência da República, segundo consta a seguir:

> Tecnologia Assistiva é uma área do conhecimento, de característica interdisciplinar, que engloba produtos, recursos, metodologias, estratégias, práticas e serviços que objetivam promover a funcionalidade, relacionada à atividade e participação de pessoas com deficiência, incapacidades ou mobilidade reduzida, visando sua autonomia, independência, qualidade de vida e inclusão social.
>
> (CAT, 2007).

Através da aplicação de ferramentas metodológicas presentes na área de Engenharia de *Software* desenvolveu-se um projeto que atendesse os requisitos necessários para um programa computacional relacionado à Tecnologia Assistiva. Esta metodologia, aplicada de maneira educativa neste trabalho, pode ser exportada para projetos de maior porte e complexidade na

indústria, sendo uma experiência válida para experiências futuras de profissionais da Engenharia.

O projeto de *software* proposto visa controlar um dispositivo robótico, um carrinho, por meio da varredura automática e com o uso de um dispositivo acionador, com a mesma funcionalidade do botão direito do *mouse*.

No Capítulo 2 são apresentados os objetivos do projeto.

No Capítulo 3 são apresentados os tópicos estudados incluindo os principais métodos de desenvolvimento de projetos, de documentação, da linguagem de programação e comunicação com *hardware.*

No Capítulo 4 são apresentados os materiais e métodos e no Capítulo 5 os resultados e discussões.

As conclusões e propostas para trabalhos futuros são apresentadas no Capítulo 6 e no Capítulo 7, as referências.

No Capítulo 8 são apresentados os anexos com os códigos utilizados na programação do *software*.

2 OBJETIVOS

O objetivo do trabalho é a produção de um projeto de desenvolvimento de um software voltado para crianças com deficiências motoras que possibilite o controle de um dispositivo robótico à distância, por meio de varredura automática e periféricos adaptados. Embora sua finalidade seja incialmente ser uma ferramenta a ser aplicada em ambiente infantil, seus recursos e funcionalidades podem ser expandidos para quaisquer outras faixas etárias.

Com este dispositivo, almeja-se a construção de uma ferramenta acessível, para que crianças sem mobilidade nos membros superiores possam desenvolver atividades didáticas, lúdicas, com autonomia. Para crianças com mobilidade reduzida, além de atividades didáticas e lúdicas, o software também pode ser utilizado como atividade de reabilitação.

Quanto aos objetivos acadêmicos, o projeto visa aprimorar os conhecimentos em técnicas avançadas de programação e estudar os principais modelos de desenvolvimento de projetos de *software*, conteúdos que não são abordados na grade curricular das disciplinas de graduação, trazendo uma experiência que pode servir de base para consultas em outras áreas além da acadêmica, como a industrial por exemplo.

3 REVISÃO BIBLIOGRÁFICA

3.1 EDUCAÇÃO INCLUSIVA DE PESSOAS COM DEFICIÊNCIAS

Avaliações concretas do número de pessoas com deficiência fazem parte de medidas relativamente novas. No Brasil, apenas a partir de 1990 seu censo demográfico (através da Lei Federal 7853, art. 17) passou a incluir questões acerca de pessoas portadoras de deficiência no país, ou seja, até então não existiam registros exatos sobre a quantidade de brasileiros com esta problemática. No panorama atual, de acordo com os dados do censo de 2010 o Brasil conta com aproximadamente 24% de sua população que se declara com algum grau de dificuldade em enxergar, caminhar, ouvir e/ou subir degraus. Já entre os que se declaram com grande ou total dificuldade nestas habilidades representam um total de 6,7% da população, cerca de 12,5 milhões de brasileiros (IBGE, 2010).

O conceito de inclusão social recorre naturalmente à educação e, para que pessoas com deficiência possam estar dentro de um ambiente escolar adequadamente estruturado para sua formação, são necessárias mudanças no projeto pedagógico, nos materiais didáticos e, por fim a mudança de paradigmas da sociedade (SISSON, 2009).

Assim, na década de 90 surge o conceito de Inclusão Social, o qual propôs a mudança de paradigmas sociais e da educação. Neste contexto, os princípios da inclusão são a valorização de cada indivíduo, a aceitação das diferenças, a diversidade, e a aprendizagem através da cooperação, enquanto que nos conceitos destacam-se a independência, a autonomia e a igualdade nas oportunidades (BRASIL, 1994). De acordo com este novo modelo social de deficiência, o problema está na sociedade e não na deficiência.

A deficiência passa então a ser encarada como um desafio a ser superado, deixando de ser sinônima de incapacidade e as instituições passam a ser desafiadas a criarem serviços e programas para atender a todos (BRASIL, 1994).

3.1.1 Recursos Tecnológicos a Serviço da Inclusão de PCD

Cabe a correta definição do termo "Tecnologia Assistiva" no campo educacional como uma miríade de recursos de acessibilidade para diferentes estudantes, com vistas a minimizar barreiras e comprometimentos advindos de uma deficiência, incapacidade ou mobilidade reduzida (GALVÃO FILHO, 2016). De fato, a TA contempla uma vasta gama de recursos, que englobam também a área plenamente de saúde, minimizando os efeitos da deficiência para a rotina de seus usuários, recreação e esportes, transporte em veículos, adaptações de ambiente,

entre outros, assim complementando a definição dada anteriormente pelo Comitê de Ajudas Técnicas.

A partir do início do século XXI, com a popularização do uso de computadores e Internet surgiu a necessidade de procurar inciativas para inserir as PCDs no processo de inclusão digital. Para este fim, encontram-se atualmente no mercado dispositivos e periféricos de ampla variedade com o intuito de minimizar as dificuldades impostas pela deficiência para o acesso a computadores. Estes dispositivos variam de acordo com o tipo de deficiência do usuário e sua respectiva gravidade e dentre eles, destacam-se as pranchas de comunicação digital, acionadores e dispositivos de controle computacional através de movimentos de partes diversas do corpo humano, como pés, cabeça e olhos, entre outros recursos.

A utilização de TA é convencionada também a um estudo de cada caso. Tendo como exemplo uma equipe especializada na avaliação, disponibilidade e indicação de recursos de TA, a "Assistiva – Tecnologia e Educação" possui uma metodologia bem estabelecida para utilização e recomendação dos recursos apropriados que possibilitem a melhor resposta para diferentes usuários (BERSCH e TONOLLI, 2006). Ou seja, o uso da TA deve ser estudado discriminadamente levando em consideração não apenas quais recursos de hardware sejam os mais adequados fisicamente, mas também quais atividades serão melhor exploradas pelo usuário.

Visto que este trabalho terá ênfase em um recurso de *software* de TA acerca da acessibilidade no uso de computadores, é válido destacar o funcionamento dos recursos destinados a este fim. O tipo de recurso utilizado é classificado como de acesso indireto, em que o usuário realiza de forma indireta e autônoma o controle do computador (BERSCH, 2013).

3.1.2 Varredura Automática

Este sistema identifica e seleciona automaticamente elementos que podem ser ativados pelo usuário do computador (BERSCH, 2013). Os sinais podem ser mostrados visualmente como forma de moldura colorida, preenchimento da área em que o elemento está contido ou em forma de áudio por um sinal que indica qual elemento está sendo selecionado. Sendo assim, através de periféricos adaptados, acionadores, ou outros hardwares de TA de acordo com a deficiência do usuário, este ativará o comando em conformidade com a seleção que passará pela varredura.

3.2 ENGENHARIA DE SOFTWARE

A Engenharia de Software é um conjunto de técnicas e ferramentas que surgiu para tentar solucionar os problemas gerados pela "Crise do software", que surgiu na década de 60 quando se iniciaram as produções de softwares de médio e grande porte.

Pressman (2012) afirma que a engenharia de *software* é uma tecnologia em camadas, onde a camada mais baixa, base de toda a abordagem presente na Engenharia como um todo, é um foco organizacional voltado à qualidade.

Especificamente sobre a Engenharia de *Software* sua base consiste nos processos a ela relacionados, onde se estabelece a metodologia a ser aplicada e definem-se os seus respectivos métodos técnicos.

A penúltima camada é composta pelos métodos, onde se realiza a análise de requisitos, modelagem do projeto, criação do programa e eventuais testes.

Por fim a camada mais superficial, tomando as anteriores como base, é a de ferramentas, onde se tem o suporte de desenvolvimento, podendo ser automatizado ou semi-automatizado. Através destes preceitos desenvolveu-se o *software* presente neste trabalho (PRESSMAN, 2012).

3.2.1 Qualidade de *Software*

A produção de elementos com qualidade é um fato que pode remitir até uma certa redundância, mas cabe uma definição mais concreta do que seria qualidade, em específico dentro da engenharia de *software*. Para Pressman (2012), a qualidade de *software* é definida como "uma gestão de qualidade aplicada de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e o utilizam".

Desta definição, destaca-se ainda que a gestão de qualidade efetiva estabelece infraestrutura necessária para a construção do produto. Em situações mais complexas é esta gestão que minimiza possibilidades de desorganização nos processos, através de ferramentas de controle dos mesmos. O produto útil, além de satisfazer os requisitos do cliente, pode trazer funcionalidades para melhor praticidade no uso e alta confiabilidade com relação a possíveis erros. Já o valor mensurável para aqueles que o produzem e o utilizam traduz-se como menor manutenção para o produtor, menores paradas para o cliente junto a uma maior agilidade na produção.

A qualidade ainda possui classificações implícitas, ou fatores, nomenclatura variável que depende do autor que as cunha. David Garvin (1988) define oito dimensões de qualidade,

que podem ser aplicadas também na engenharia de *software*. Já McCall, Richards e Waters (1977) propõem até 11 fatores que afetam a qualidade do *software*. De maneira mais detalhada, cita-se seis fatores de qualidade presentes no padrão ISO/IEC 9126, que é uma norma voltada para qualidade de *software*, sendo elas:

- Funcionalidade: capacidade que o *software* tem de atender às devidas necessidades ao ser utilizado em condições específicas, incluindo adequação, acurácia, interoperabilidade e segurança.
- Confiabilidade: capacidade de o *software* manter um nível de desempenho especificado previamente sob condições específicas, incluindo maturidade, tolerância a falhas e recuperabilidade.
- Usabilidade: capacidade do *software* se mostrar prático, de fácil aprendizagem e operação ao usuário, incluindo inteligibilidade, apreensibilidade, operacionalidade e atratividade.
- Eficiência: capacidade do *software* em apresentar desempenho apropriado sem requerer recursos demasiados sob condições específicas, incluindo comportamento em relação ao tempo, como tempo de resposta e processamento e utilização de recursos como memória dinâmica.
- Manutenibilidade: capacidade do *software* de ser modificado, corrigido ou adaptado devido a mudanças em seu ambiente de aplicação e/ou nos seus requisitos, incluindo analisabilidade, modificabilidade, estabilidade e testabilidade.
- Portabilidade: capacidade do *software* de sofrer mudanças de ambiente sem perder suas funcionalidades, incluindo capacidade de instalação, coexistência em variados ambientes e capacidade para substituição.

Pressman (2012) afirma que estes fatores descritos pela ISO/IEC 9126, ou mesmo os fatores dados por McCall ou as dimensões de Garvin não proporcionam necessariamente uma medição direta. Contudo, medidas indiretas podem ser tomadas através destas bases sendo assim um ponto de partida para uma análise da qualidade de um sistema.

3.2.2 Processos de *Software*

Define-se como processos de software o conjunto de atividades intrínsecas à produção do produto (SOMMERVILLE, 2011). Assim como a qualidade, os processos são atividades que podem ser realizadas de variadas maneiras, mas para a área de engenharia de software são necessárias no mínimo: especificação do *software*, projeto e implementação, validação e

evolução. Para além destas atividades básicas, podem ser realizadas outras como atividades de complementação e apoio ao processo.

Pressman (2012) amplia esta definição para, além de atividades, o processo também se constituir de ações e tarefas na construção do produto de *software*, onde tarefa tem como objetivo um resultado de menor grau e concreto dentro do processo, enquanto que ação seria um conjunto de tarefas que formam um artefato de *software* fundamental, como um modelo de projeto de arquitetura.

Ou seja, apesar do processo possuir certa flexibilidade com relação a sua aplicação, a maior parte dos casos requer uma metodologia, também chamada de *framework*, envolvendo desde a comunicação entre cliente e desenvolvedor, um planejamento do projeto, modelagem, construção e emprego do *software* (PRESSMAN, 2012).

A comunicação consiste na etapa incial realizada entre cliente e a equipe desenvolvedora. Esta se mostra essencial para se entender quais os requisitos e principais necessidades que o produto final irá atender e/ou quais destes requisitos são alcançáveis e suas complexidades.

O planejamento define quais tarefas técnicas serão realizadas e por quem dentro da equipe, além de estabelecer cronogramas para as atividades, recursos necessários para a realização das mesmas, assim como custos e riscos do projeto.

Na modelagem são realizados esboços do *software* de acordo com as necessidades e requisitos dados previamente no planejamento. Estes esboços podem possuir nível de detalhe variável de acordo com a complexidade do projeto e possuem sintaxe bem definida, que abordados com maiores detalhes nas próximas seções.

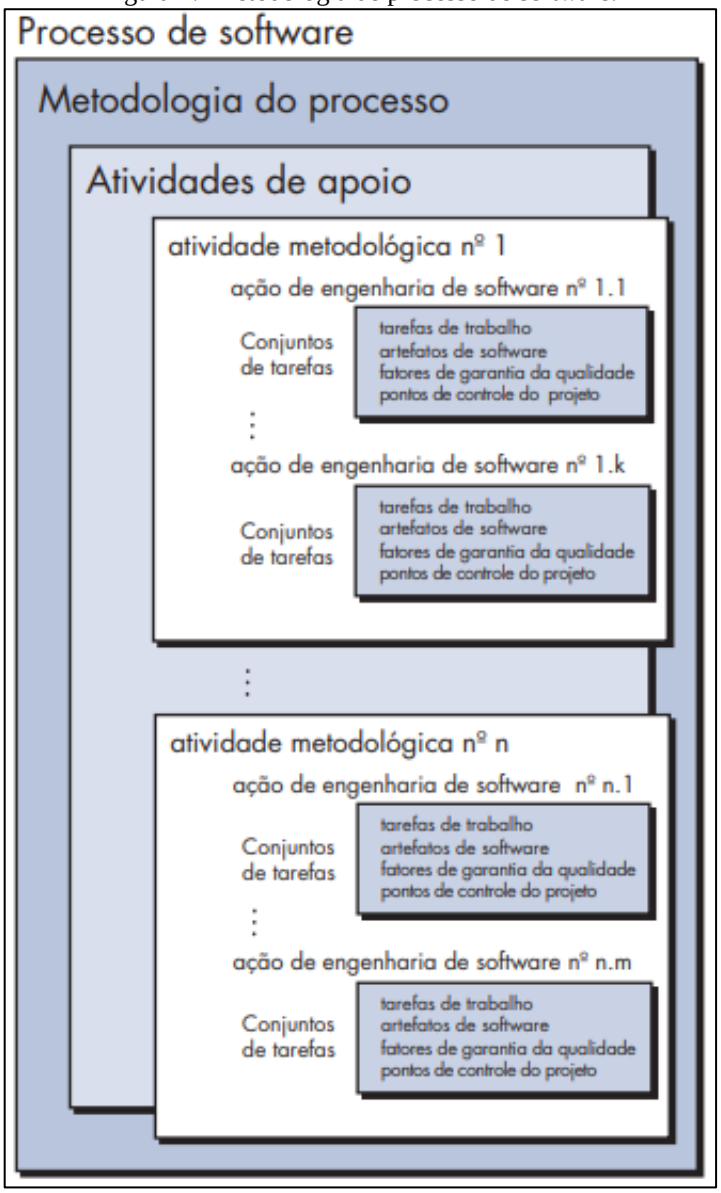
Já a construção é efetivamente a realização da codificação e testagem, levando em consideração a modelagem feita anteriormente.

Por fim, o emprego é a efetiva entrega do produto final, com avaliação feita pelo usuário e posterior revisão e tratamento de erros de acordo com seu parecer.

Um diagrama das etapas do processo é apresentado na Figura 1, onde cada atividade metodológica (comunicação, planejamento, modelagem, construção, emprego e/ou outras adicionais) são perpassadas por atividades de apoio dentro processo, como acompanhamento e controle, administração de riscos e recursos, entre outras. Ressalta-se que estas atividades básicas não precisam ser executadas necessariamente em ordem linear. De acordo com as necessidades tanto dos desenvolvedores, quanto para o mercado no qual o produto final está inserido ou até mesmo os requisitos do cliente, as atividades podem ser realizadas de forma

iterativa, onde uma ou mais atividades podem se repetir mais de uma vez, ou ocorrerem em paralelo umas com as outras, isto é definido previamente pela equipe de desenvolvimento ou no decorrer do processo.

Figura 1: Metodologia de processo de software.



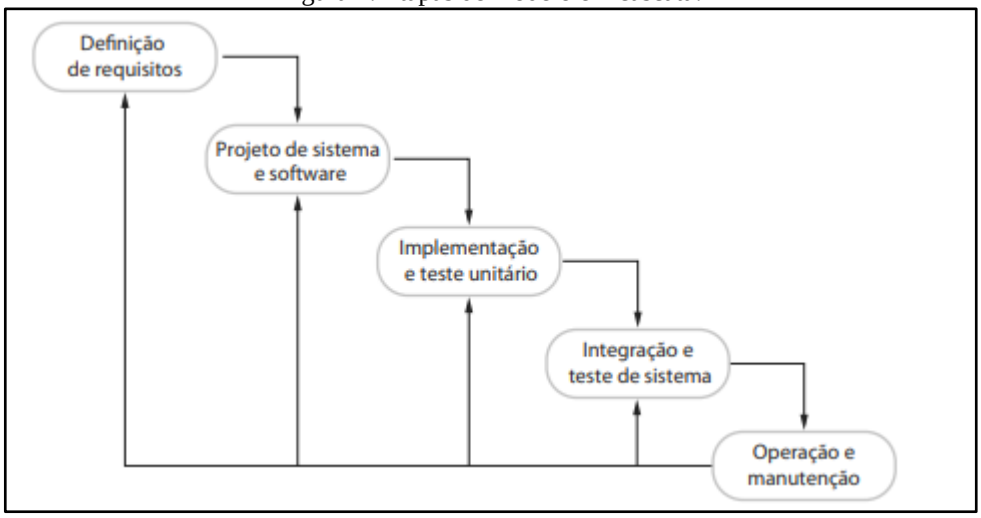
Fonte: (PRESSMAN, 2012).

Em muitas situações são encontrados problemas ao longo do processo de *software* e para resolvê-los a equipe de desenvolvimento, por meio de um conjunto de soluções pré-estabelecidas, pode se aproveitar de padrões de processos já existentes. Ou seja, utilizar métodos de solução de problemas a partir de modelos bem estabelecidos, e com a combinação de diversos padrões a equipe poderá prever reações para eventuais problemas que surjam a partir destes modelos genéricos.

Através dos padrões de processo citados anteriormente, pode-se usar modelos de processo prescritivo. Um modelo de processo pode mostrar uma visão ampla de todo o processo ou apenas uma perspectiva parcial. Destacam-se três tipos principais de modelos de processo que podem ser tomados: Modelo Cascata, Modelo Incremental e Modelo de Processo Evolucionário.

Modelo em cascata: modelo cuja representação de exemplo encontra-se na Figura 2, constitui-se de um encadeamento entre as fases, em que as atividades metodológicas são realizadas de maneira sequencial. Neste caso, a definição de requisitos classifica-se de maneira mais generalizada como comunicação. Já o projeto de sistema e *software* junto à implementação e teste unitário fazem parte das atividades de planejamento, modelagem e construção, e, por fim, integração e teste de sistema assim como operação e manutenção estão englobados na atividade metodológica "emprego".

Figura 2: Etapas de modelo em cascata.

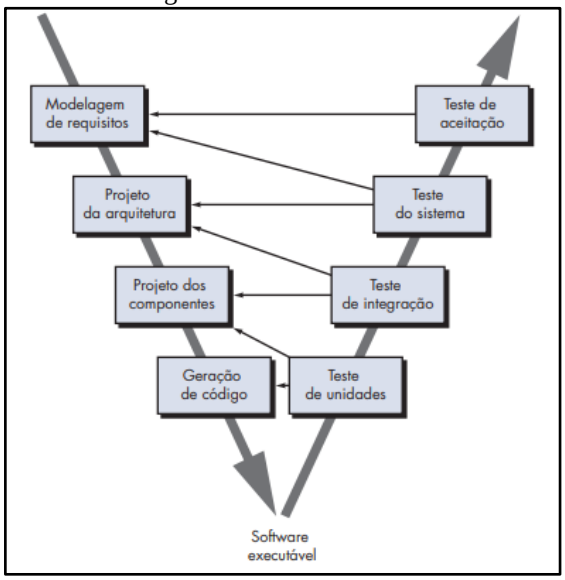


Fonte: (SOMMERVILLE, 2011).

Este modelo é indicado quando os requisitos são bem definidos, assim como a integração e manutenção também o são, podendo ser como exemplo atualizações ou adaptações de sistemas já existentes. Apesar de ser o modelo mais antigo presente na engenharia de *software*, tem sua eficácia questionada na área, muito pela raridade de se obter requisitos tão claros e objetivos, a necessidade de se chegar a estágios mais avançados antes de obter as primeiras versões operacionais e a resistência à eventuais mudanças no meio do processo (PRESSMAN, 2012).

O Modelo em "V", representado na Figura 3, é uma variação do modelo em cascata no qual a partir da geração do código, as etapas a montante da geração do executável são transpassadas por uma série de testes, garantindo ações de qualidade.

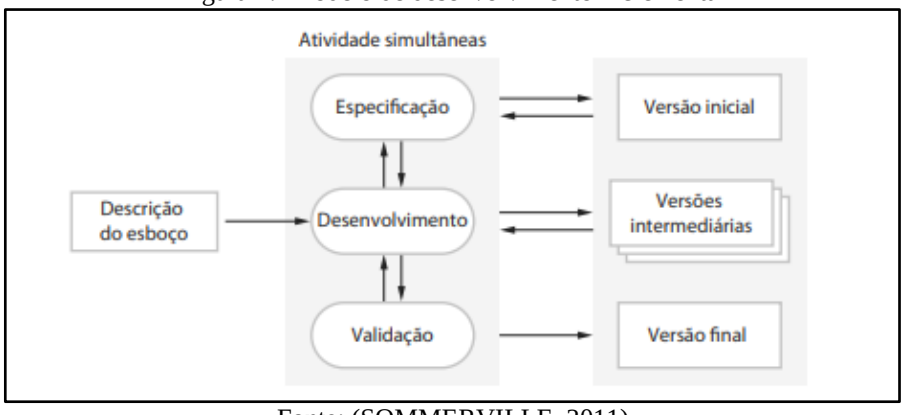
Figura 3: Modelo em "V".



Fonte: (PRESSMAN, 2012).

Modelo incremental: sua representação encontra-se na Figura 4. Constitui-se de um desenvolvimento de uma primeira versão funcional para avaliação do cliente, para então construírem-se novas versões com melhorias e novas funcionalidades, sempre passando pela entrega e avaliação antes do desenvolvimento da próxima (SOMMERVILLE, 2011). As atividades metodológicas presentes no desenvolvimento de cada versão podem ser lineares ou paralelas, de acordo com requisitos, mudanças de projeto ou outras eventualidades. Sendo assim, pode-se focar em expandir suas funcionalidades sem comprometer prazos e/ou a base para seu funcionamento.

Figura 4: Modelo de desenvolvimento incremental



Fonte: (SOMMERVILLE, 2011)

Sua primeira versão normalmente possui apenas ferramentas básicas para o seu funcionamento, ainda sem possuir todos os requisitos originais complementares. É apenas com o desenvolvimento de novas versões que estes serão atendidos. Salienta-se também que é feito

um planejamento adequado para cada incremento seguinte, de acordo com o resultado da avaliação feita pelo cliente da versão incremental anterior.

Comparativamente ao modelo em cascata, o modelo incremental exige menos recurso de pessoal, por exemplo, uma menor equipe pode desenvolver a primeira versão funcional e receber auxílio para as atualizações posteriores (PRESSMAN, 2012). O retorno do usuário pode também garantir uma maior clareza nos requisitos ao se obter as primeiras versões e, por fim, o software pode ser entregue com funcionalidades de maior uso e urgência mais rapidamente, sendo uma vantagem para o cliente dependendo do contexto onde será utilizado.

Segundo Sommerville (2011) este tipo de desenvolvimento de processo é a abordagem comum, porém encontra desvantagens na aplicação em longo prazo ou para sistemas com grande robustez, pois estes exigem arquitetura estável e várias equipes necessitam desenvolver partes diferentes do produto. Assim, o respeito a esta arquitetura se mostra mais dificultoso, ainda mais se existir correlação entre as partes desenvolvidas, a sincronia entre as equipes deve ser de tal maneira que pode este modelo se tornar de difícil aplicação.

Modelos de processo evolucionários: São modelos de processo mais adequados para mudanças constantes de requisitos e prazos. Quando se mostra inadequado montar um planejamento linear para o processo, necessita-se de um modelo que permita um desenvolvimento contínuo do software ao longo do tempo. Assim são classificados os modelos evolucionários, com destaque às suas duas vertentes mais comuns: a prototipação e o modelo espiral.

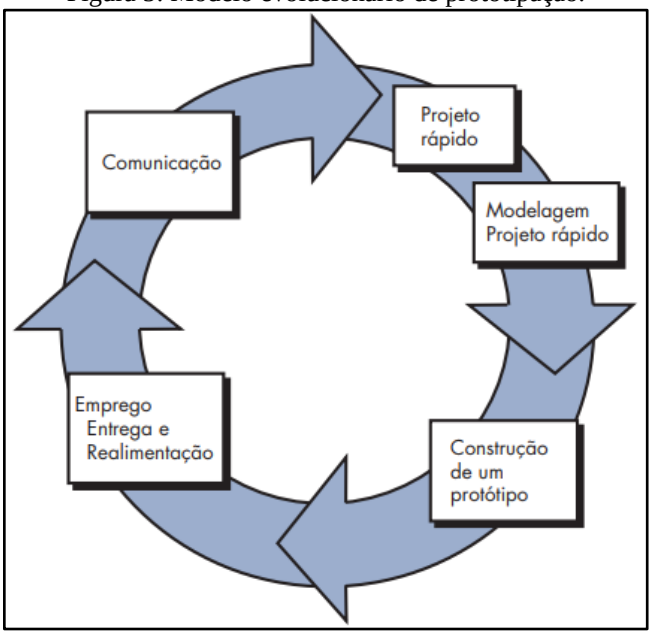
Prototipação: Mais do que um modelo de processo, a prototipação já pode ser classificada também como uma técnica podendo ser implementada em qualquer outro modelo (PRESSMAN, 2012). Caracteriza-se como uma formação de um protótipo através da apresentação dos requisitos mais gerais dados pelo cliente e funcionalidades necessárias para sua execução inicial. É uma abordagem bem utilizada quando os requisitos não são claramente especificados, os objetivos mais gerais, ou quando o desenvolvedor não sabe com clareza sobre aspectos de adaptabilidade com o ambiente, como se dará a interação homem/máquina ou à própria eficiência do algoritmo (PRESSMAN, 2012).

A comunicação com o cliente é feita de maneira a se apresentar um esboço de elementos visíveis ao usuário como layout de interface por exemplo. Sendo assim, a comunicação, planejamento e modelagem são feitos rapidamente tendo como resultado um protótipo para avaliação e feedback do usuário, em busca da compreensão mais clara dos requisitos. Ocorre então uma iteração, atualizando o protótipo de acordo com o entendimento mais adequado do

que se tem como necessidade. Este protótipo pode ser operacional, sendo evoluído posteriormente, mas pode também ser uma versão descartável e inutilizável, apenas servindo para melhor compreensão dos requisitos.

Na Figura 5 a seguir apresenta-se uma abordagem de prototipação:

Figura 5: Modelo evolucionário de prototipação.



Fonte: (PRESSMAN, 2012).

Apesar de ser um paradigma eficiente, Pressman (2012) alerta para possíveis problemas da prototipação. A apresentação de um protótipo semelhante a uma versão final pode levar o cliente a tê-lo como produto, ou que peça que o produto final tenha apenas algumas alterações, sem levar em conta a qualidade ao longo prazo, aplicação global de qualidade, portabilidade, entre outros métodos de qualidade ainda não verificados ou aplicados. O propósito de se operacionalizar com rapidez o protótipo pode também levar ao desenvolvedor assumir riscos desenvolvendo o software de acordo com as condições específicas do momento atual em que este será aplicado, sem atualizá-lo posteriormente.

Recomenda-se então utilizar o protótipo para definição com maior clareza dos requisitos necessários, com todos os envolvidos sabendo de início que este é o motivo pelo qual ele está sendo apresentado, e que o software em si será produzido com base em preceitos de qualidade ainda não aplicados, com o protótipo em si ou parte dele podendo ser descartado.

Modelo espiral: Proposto por Barry Boehm (1988), este modelo, ilustrado na Figura 6, aplica conceitos de prototipação em aplicações mais lineares do modelo em cascata. Seu início consiste na determinação de objetivos, alternativas e restrições, passando pela análise de riscos, formação do primeiro protótipo, sua modelagem, análise e simulação, para então apresentá-lo

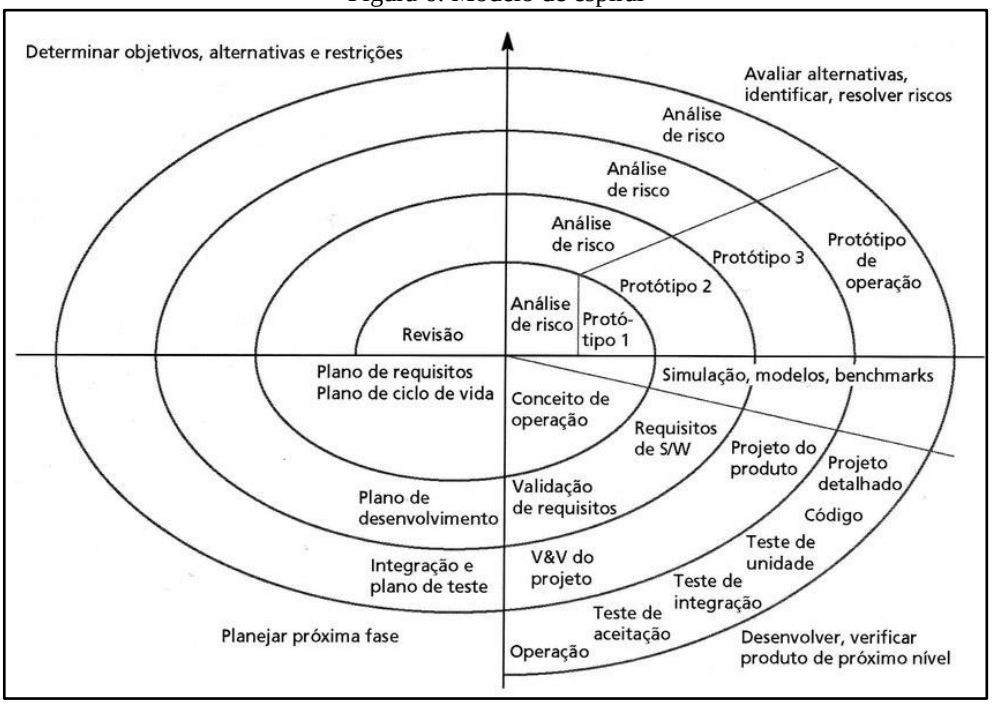
ao cliente e obter com maior clareza os requisitos. Após a primeira fase, planeja-se o desenvolvimento de um novo protótipo, definindo seus requisitos de *software*, repetindo-se os passos anteriores. A cada volta completa na espiral definem-se os requisitos com maior clareza, planeja-se o desenvolvimento do *software* com mais robustez e complexidade e apresenta-se um novo protótipo para maiores atualizações.

O eixo das abscissas é caracterizado pelo progresso conseguido a cada ciclo da espiral e o das ordenadas seu custo incorrido acumulado. Seu primeiro quadrante é caracterizado pela análise de riscos e formação de soluções e alternativas. No segundo quadrante determinam-se objetivos gerais e requisitos. No terceiro ocorre o planejamento das próximas etapas e no quarto o desenvolvimento, verificação para atualizações no produto e a modelagem, simulação e análise, com o fluxo de atividades seguindo no sentido horário.

Observa-se que para cada volta na espiral, um detalhamento maior do processo é conseguido, mas seu custo também é aumentado, e que obviamente um maior progresso ocorre no produto final do que nos seus objetivos e requisitos.

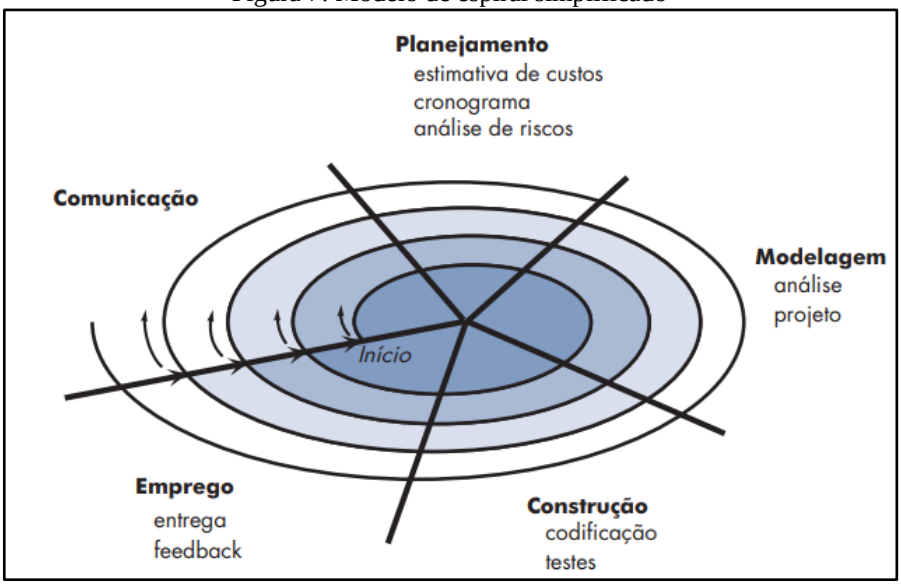
De maneira mais simplificada, visando maior entendimento e praticidade do modelo para utilização pela equipe de desenvolvimento, apresenta-se também na Figura 7 um exemplo de menor detalhamento da espiral, com apenas as atividades metodológicas mais básicas já apresentadas anteriormente.

Figura 6: Modelo de espiral



Fonte: (BOEHM, 1988).

Figura 7: Modelo de espiral simplificado



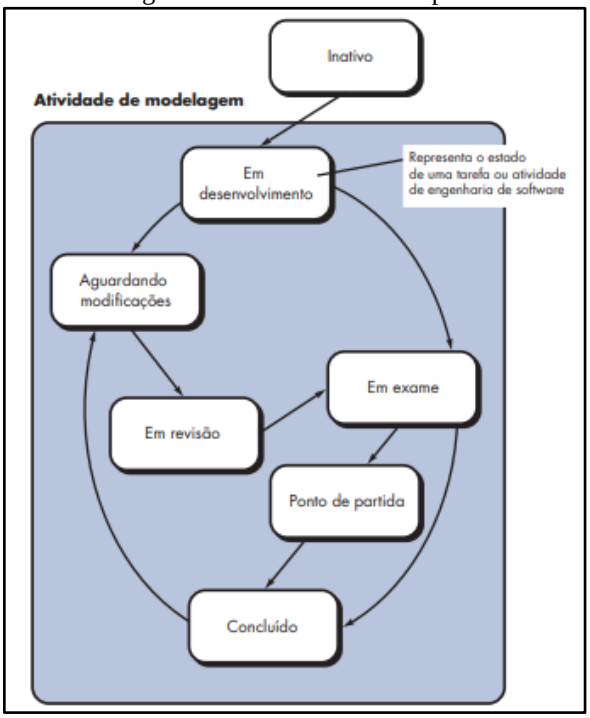
Fonte: (PRESSMAN, 2012).

Este modelo possibilita uma contínua atualização do *software* com o passar do tempo. Com seu aperfeiçoamento ao longo do tempo, cliente e desenvolvedor podem compreender melhor os riscos e requisitos, assumindo uma sistemática mais linear de acordo com o modelo em cascata, mas utilizando elementos mais compatíveis com situações reais em larga escala, através da prototipação e iteração. Suas desvantagens consistem-se na exigência de constante avaliação de riscos e caso isto não ocorra de maneira minuciosa, problemas poderão ocorrer na entrega do produto.

Por fim uma análise traçada por Nogueira (2000) destaca algumas problemáticas dos modelos evolucionários, segundo ele "os processos de evolucionários não estabelecem uma velocidade máxima da evolução". Assim uma evolução rápida demais pode provocar desorganização no seu desenvolver, ou então uma velocidade muito lenta poderia levar a perda de produtividade. Destaca-se também que o foco total na qualidade pode trazer uma demora no prazo de entrega do produto e eventual perda de mercado dependendo do contexto, e é a busca pela alta qualidade o principal objetivo dos modelos de processo evolucionários.

Abordagem de modelos concorrentes: Consiste em uma modelagem mais genérica para atividades metodológicas presentes no processo. Diferentemente dos modelos descritos anteriormente, o modelo de desenvolvimento concorrente possibilita uma análise instantânea do processo, de acordo com a situação de cada atividade, como mostra a figura 8 a seguir.

Figura 8: Elemento genérico de um modelo de processo concorrente



Fonte: (PRESSMAN, 2012).

Sobre o modelo de processo concorrente, Pressman (2012) faz a seguinte afirmação: "A modelagem concorrente define uma série de eventos que irão disparar transições de estado para estado para cada uma das atividades, ações ou tarefas da engenharia de *software*."

Sendo assim, modelos concorrentes consideram a realização de diversas atividades simultaneamente, e quais eventos provocam uma mudança de estado para cada uma delas.

Metodologias Ágeis: são voltadas para desenvolvedores e programadores experientes. Tem como principais características: priorizar as pessoas e interações, ao invés de processos e ferramentas; produzir um software executável, ao contrário de documentação extensa e confusa; trabalhar diretamente com colaboração do cliente, ao contrário de constantes negociações de contratos e obter respostas rápidas para as mudanças, ao contrário de seguir planos previamente definidos (PRIKLADNICKI et al., 2014). Entre as principais diferenças da XP em relação às Metodologias Clássicas estão o feedback constante, a incremental e o encorajamento da comunicação entre as pessoas.

Os mais difundidos são o *Extreme Programming*, voltada para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente e o método *Scrum*, que se concentra principalmente no gerenciamento de tarefas dentro de um ambiente de desenvolvimento baseado em time. Ele é relativamente simples de implementar e aborda muitos dos aspectos complexos de gestão. Este método impõe disciplina que permite um acompanhamento mais próximo do andamento do projeto.

3.2.3 Processo Unificado

A presença de diversos modelos de processo com características diferentes e para diversas aplicações levou ao desenvolvimento de um padrão de processo unificado (PU) que possuísse características dos principais modelos propostos até então. A partir dos anos 90, criou-se um padrão para modelagem no desenvolvimento de softwares: A UML – Linguagem de Modelagem Unificada.

A partir da criação da UML se desenvolveu um padrão de processo unificado, proposto por Rumbaugh, Booch e Jacobson (1995), uma metodologia de processo para aplicação em engenharia de software.

O PU propõe cinco fases relacionadas às atividades metodológicas básicas, representadas na Figura 9 presente na página a seguir.

A fase de concepção engloba as atividades de comunicação com o cliente e planejamento, onde se identificam de maneira geral objetivos, requisitos necessários, contexto de mercado no qual o produto estará inserido, riscos e recursos a serem utilizados, propõe a criação de uma arquitetura operacional para auxílio na identificação de requisitos, além do cronograma inicial de trabalho.

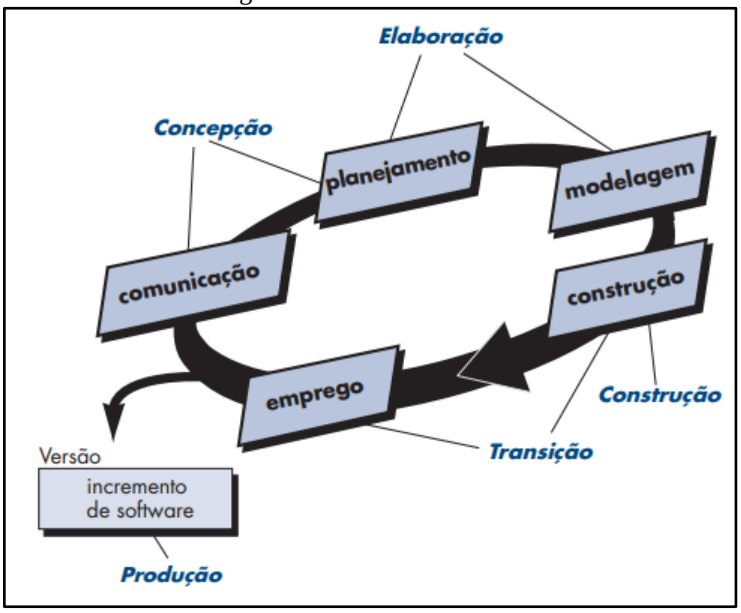
A fase de elaboração engloba atividades de planejamento e modelagem, onde ocorre um refinamento da arquitetura inicial. Esta arquitetura pode ou não ser um sistema executável. Ocorre também uma atualização de cronogramas e análises feitas na concepção.

A fase de construção é muito semelhante à atividade metodológica de construção, de maneira a desenvolver o modelo de arquitetura a incluir aspectos e componentes de software (PRESSMAN, 2012), tornando-o operacional, ou seja, conclui-se o desenvolvimento de cada incremento através da implementação das funções requeridas, realizando-se a implementação e testes do mesmo.

A fase de transição integra as últimas atividades de construção e emprego, o usuário faz a avaliação das primeiras versões e gera um feedback que será utilizado no retorno da fase de concepção, para posterior desenvolvimento de novos incrementos.

Ao se chegar na versão final, entra em ação a fase de produção em que se é gerada a partir da atividade de emprego: o software é implementado e seu uso acompanhado continuamente, com o devido suporte disponibilizado para o usuário. Estas cinco fases não são necessariamente sequenciais e podem ocorrer concomitantemente (PRESSMAN, 2012).

Figura 9: Processo unificado



Fonte: (PRESSMAN, 2012).

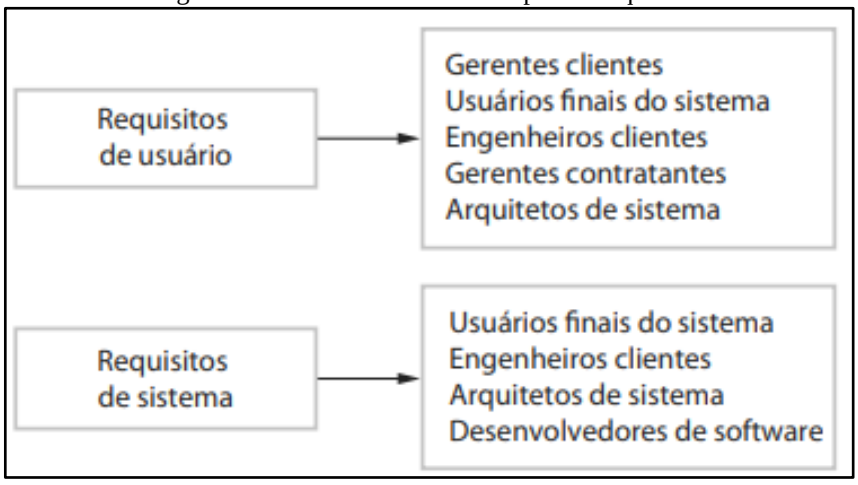
3.2.4 Engenharia de Requisitos

Define-se como requisito de software qualquer descrição de uma necessidade vinculada ao seu projeto. Esta necessidade pode estar relacionada aos seus serviços, funcionamento ou restrições de uso. Sommerville (2011) faz uma diferenciação entre dois tipos diferentes de requisitos: os que contêm maior nível de abstração, constituídos de declarações de serviços que o sistema deve oferecer ao usuário e restrições de uso são chamados de requisitos de usuário; já os descritos de maneira mais trabalhada, com funções, serviços e restrições operacionais com nível mais baixo de abstração são chamados de requisitos de sistemas.

Como citado anteriormente, na maioria dos casos estes requisitos não são definidos claramente no início do projeto e, além disso, podem sofrer alterações com o desenvolvimento do processo em si. Para auxiliar os desenvolvedores a obter requisitos de maneira mais clara formulou-se então a engenharia de requisitos, com metodologias específicas com vistas a este objetivo. Relembrando as atividades metodológicas básicas, os requisitos são um dos principais alvos da comunicação, de tal maneira que seu detalhamento é feito em diferentes níveis de abstração de acordo com seu comunicador, sendo usuário, cliente ou desenvolvedor, como representado na Figura 10.

Sobre a comunicação entre as partes interessadas, desenvolvedores e clientes com os últimos sendo desde usuários finais, gerentes de sistemas e até reguladores ou outros agentes externos interessados, realizam-se descrições gerais dadas pelos clientes, documentação, entrevistas e reuniões, apresentação de protótipos pelos desenvolvedores etc.

Figura 10: Leitores de diferentes tipos de requisitos



Fonte: (SOMMERVILLE, 2011).

3.2.5 Padrão UML

A UML é um padrão desenvolvido para descrição, construção e visualização de sistemas de *software*. Como afirmam Rumbaugh, Booch e Jacobson (2006) a UML é uma linguagem, e, assim como qualquer outra, possui regras de sintaxe e vocabulário próprios. Ela pode ser aplicada desde a documentação da arquitetura do sistema, expressão de requisitos, realização de testes e planejamento de processo (através do PU).

Os blocos de construção mais básicos da UML são compostos por três tipos: os itens, os relacionamentos e os diagramas, onde os itens possuem nível de composição mais baixo, os relacionamentos são interligações entre itens e diagramas são agrupamentos de itens relacionados entre si.

Existem quatro tipos de itens presentes na UML, sendo o único tipo utilizado neste trabalho o item estrutural. Os outros três, sendo eles itens comportamentais, itens de agrupamentos e itens anotacionais não sendo utilizados e, portanto, para efeito de menor prolixidade, não serão abordados nesta seção.

Itens estruturais: São componentes estáticos, representam elementos conceituais ou físicos (RUMBAUGH, BOOCH, JACOBSON, 2006), compostos por classes, interfaces, casos de uso, entre outros.

Classes: Descrições de um conjunto de objetos compartilhando os mesmos atributos, operações, relacionamentos e semântica. São indicadas por retângulos, incluindo nome, atributos e operações.

Interfaces: São implementadas pelas classes, descrevem o comportamento externamente visível de um determinado elemento, podem descrever o comportamento total ou

parcial de uma classe. Indicam-se de maneira semelhante às classes, mas com uma *tag* ≪interface≫ no topo.

Atributos de itens: Itens estáticos como classes e interfaces podem possuir atributos de objetos que nela façam parte, objetos esses que podem ser variáveis ou constantes. Os atributos são propriedades dadas por estes objetos sendo eles: nome, tipo, visibilidade e valor inicial (opcional). O tipo de atributo é definido como real, inteiro, caractere, etc.

Sua visibilidade é definida como uma acessibilidade do atributo a objetos de outros itens, podendo ser: atributo privativo, com acessibilidade limitada apenas ao item na qual está inserido, representado pelo símbolo (-) antes do nome, atributo público, com acessibilidade global a todos os itens do sistema, representado pelo símbolo (+) antes do nome e atributo protegido, com acessibilidade limitada parcialmente, de acordo com algum tipo de restrição, representado pelo símbolo (#) antes do nome.

A visibilidade não precisa ser sempre explicitada, para o caso de sua representação ser omitida considera-se visibilidade do tipo privativa.

Métodos ou operações: As funcionalidades atribuídas ao item, possuem também especificações como nome, tipo de valor de retorno caso a operação o gere, lista de argumentos se a operação receber parâmetros de execução e visibilidade, explicitada da mesma maneira que nos atributos.

Para itens como classes e interfaces, delimita-se seus nomes na primeira divisão do retângulo, atributos na segunda divisão e operações na terceira.

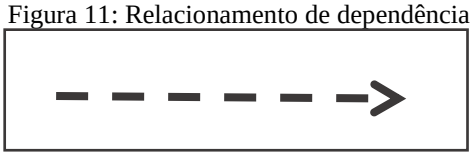
Casos de uso: Descrições de um conjunto de ações realizadas pelo sistema disparando um comportamento sequencial para outros elementos, denominados atores, normalmente através da ação de um usuário ou produtor de informação. Graficamente são representados por elipses de linha contínua. São representações que fazem parte de um nível maior de abstração do sistema, logo não possuem especificações.

Relações de casos de uso: São eventos relacionados exclusivamente aos casos de uso, podendo ser de inclusão ou extensão. As relações de inclusão indicam a obrigatoriedade da ocorrência de um caso de uso secundário quando ocorre um caso de uso primário e é representada por uma seta tracejada rotulada com o inscrito ≪*include*≫. Já as relações de extensão indicam uma possibilidade de ocorrência de um caso de uso apenas através de uma situação específica, ou se uma determinada condição for aceita. Sua representação é também indicada por uma seta tracejada, agora rotulada pelo inscrito ≪*extend*≫.

Relacionamentos: Explicitam a relação entre os itens descritos na UML. Podem ser classificados como relacionamentos de dependência, associação, generalização ou realização.

A dependência representa que a alteração de um item (independente) afeta outro (dependente). Graficamente desenhada por uma linha tracejada contendo seta no final, como indicado na Figura 11:

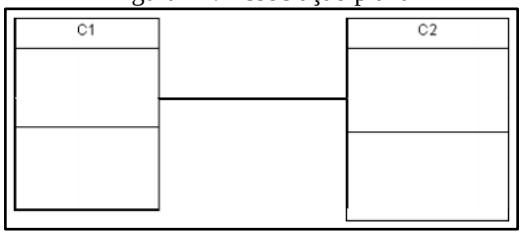
Figura 11: Relacionamento de dependência



Fonte: Próprio autor.

A associação representa a conexão estrutural entre dois itens, ou seja, demonstra que um item está incluso em outro, fazendo parte da estrutura de outro. A associação pode ser plana ou de agregação. A associação plana representa uma conexão de mesma importância entre dois itens, representada por uma linha contínua entre eles, como exemplificado na Figura 12 para classes genéricas 1 e 2:

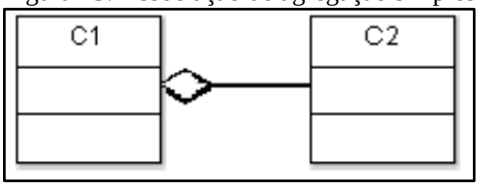
Figura 12: Associação plana



Fonte: (VIDA, 2020)

A associação por agregação divide-se em dois tipos, simples e de composição. A agregação simples representa uma associação estrutural entre dois itens que não são igualmente importantes. Graficamente é representado por uma linha contínua com um diamante aberto no final apontando o item mais importante, como mostra a Figura 13.

Figura 13: Associação de agregação simples

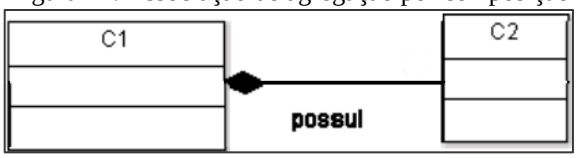


Fonte: (VIDA, 2020)

A agregação por composição indica relação forte entre itens, em que um item existe apenas através da existência do outro (mais importante). Graficamente representado por uma

linha contínua com um diamante fechado no final apontando o nível de importância, demonstrado na Figura 14 a seguir:

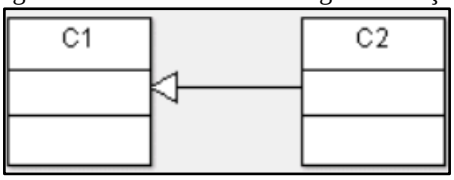
Figura 14: Associação de agregação por composição



Fonte: (VIDA, 2020)

A generalização é um relacionamento que indica que um item é uma especialização ou detalhamento de outro. Sua representação consiste numa linha contínua com uma seta fechada apontada para o item com o maior nível de generalização, como indica a Figura 15:

Figura 15: Relacionamento de generalização



Fonte: (VIDA, 2020)

Por fim, representa-se em UML a ação provida por um ator envolvido como um boneco-palito, podendo existir mais de uma ação de acordo com interligações no sistema, obedecendo a sintaxe de relacionamentos.

3.2.6 Modelagem

A modelagem de sistema em engenharia de *software* possui um papel ligeiramente diferente de sua aplicação em outros espectros da engenharia. Diferentemente de um esboço do que se terá como produto final em menor escala, o desenvolvimento de *software* necessita de uma modelagem de projeto, de processo (como mostrado na seção anterior) mas também de requisitos. Sobre a modelagem em engenharia de *software* assim afirma Pressman (2012):

> [O modelo] deve ser capaz de representar as informações que o software transforma, a arquitetura e as funções que possibilitam a transformação, as características que os usuários desejam e o comportamento do sistema à medida que a transformação ocorra. Os modelos devem cumprir esses objetivos em diferentes níveis de abstração — primeiro, descrevendo o software do ponto de vista do cliente e, posteriormente, em um nível mais técnico.
>
> (PRESSMAN, 2012).

Modelagem de requisitos: Com o intuito de ter-se uma maior clareza nos requisitos, elaboram-se modelos para este fim. Considerando diferentes cenários e contextos, produz-se diferentes tipos de modelos, com os principais dados por: modelos baseados em cenários, modelos de classes, modelos comportamentais e modelos de fluxo.

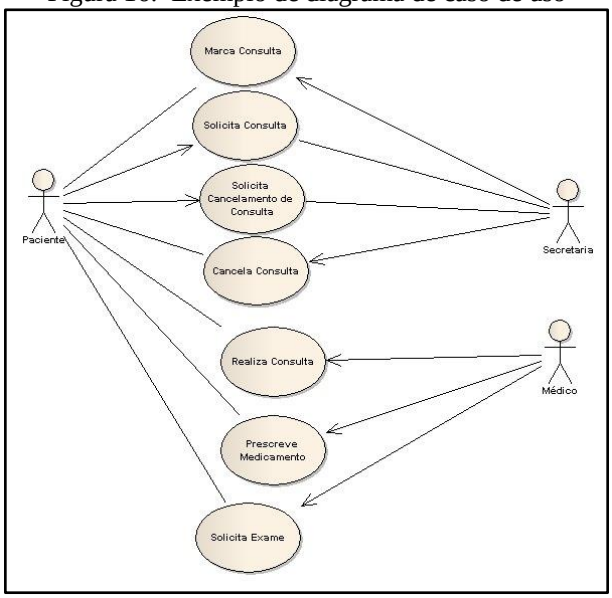
Estes diferentes tipos de modelos produzem diferentes pontos de vista sobre o sistema, cabendo à equipe de desenvolvimento escolher qual(is) destes mais se adapta(m) ao contexto no qual é realizado o projeto.

Modelos baseados em cenários apresentam uma visão de interação do usuário com o sistema; os modelos de classes representam os itens associados ao sistema, suas interações e respectivas operações; modelos comportamentais representam mudanças no sistema provocadas por agentes externos e, finalmente, modelos de fluxo representam o transporte de objetos e informações através do sistema, assim como eventuais transformações que possam sofrer. Os modelos de requisitos levam no geral a uma descrição dos requerimentos dados pelo cliente, produzem uma base para o desenvolvimento do *software* e a uma especificação dos requerimentos que o *software* possuirá para seu funcionamento adequado.

Um ponto de partida bem definido para a modelagem de requisitos pode ser baseado em cenários, a partir da construção em UML com diagramas de casos de uso, diagramas de atividades e/ou diagramas de raias (PRESSMAN, 2012).

Diagrama de caso de uso: De acordo com a definição de caso de uso dada anteriormente, monta-se um diagrama que engloba as ações e interações dentro do sistema, tendo como exemplo na Figura 16 a seguir. Ressalta-se que é essencial para o projetista que considere todas as ações possíveis que o usuário possa realizar e não apenas o que se espera incialmente, sendo estas consideradas exceções nos casos de uso, ou seja, condições que provocam comportamentos inesperados ao sistema, normalmente devido a falhas ou erros. Os diagramas de caso de uso são normalmente o ponto de partida na análise de requisitos.

Figura 16: Exemplo de diagrama de caso de uso

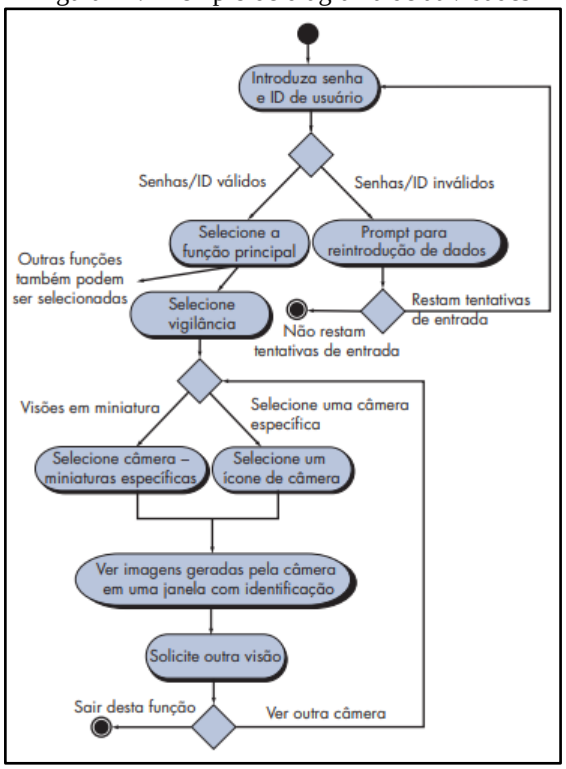


Fonte: (BOOCH, RUMBAUGH, JACOBSON, 2006).

Diagrama de atividades: Se o sistema em questão requer uma análise mais profunda e detalhada sobre as atividades a serem realizadas sobre ele, um diagrama de caso de uso pode não ser o suficiente para sua compreensão. Um diagrama de atividades, sendo um complemento do anterior, representa o fluxo destas interações para casos em que exista uma complexidade maior entre elas.

A sintaxe deste diagrama consiste em retângulos com bordas suaves representando funções do sistema, setas normais para o fluxo, losangos para decisões, círculos preenchidos em preto para início das atividades e círculos também pretos, mas com borda branca para fim das atividades. Ainda podem existir círculos em branco com "x" indicando fim do fluxo local, mas ainda podendo ocorrer atividades paralelas. Por fim as barras sólidas indicam uma sincronização, onde dois ou mais fluxos levam a uma mesma função. Um exemplo é dado na Figura 17 a seguir:

Figura 17: Exemplo de diagrama de atividades



Fonte: (PRESSMAN, 2012)

Diagrama de raias: Consiste numa variação do diagrama de atividades, aqui estas são descritos diferenciando-se o ator que as pratica, ou seja, é uma representação do fluxo de atividades descrito pelo caso de uso (PRESSMAN, 2012). O diagrama é montado em uma tabela cujas colunas fazem a divisão de atividades entre cada ator no sistema.

Modelagem de dados: A modelagem de dados é uma parte importante da modelagem de requisitos e seu objetivo é definir objetos de dados processados no sistema, ou seja, a quantidade de informação gerada e processada, vide que em muitos casos o sistema opera ou gera uma quantidade significativa de dados. Objetos de dados são expressos como representações de atributos ou propriedades de elementos, também considerados como itens em UML. Dentro desta premissa montam-se modelos de dados representando estes itens, suas interações, atributos e operações. Neste caso os modelos construídos encontram-se em níveis mais baixos de abstração, com foco maior numa leitura para os desenvolvedores, tendo como exemplo diagramas de classe.

Diagrama de classe: É a representação sistêmica de um conjunto de classes e interfaces e suas respectivas interações, de acordo com a sintaxe em UML. Com o diagrama de classe se obtém o conjunto de objetos e suas relações, como são seus atributos, interações entre si e também quais operações disparam. É um dos diagramas fundamentais para análise de requisitos.

Modelagem orientada a fluxos: Para sistemas em que o fluxo de informação é bem orientado e existem dados sendo transferidos para várias seções, modelos orientados a fluxos podem ser de utilidade interessante. De acordo com Pressman (2012) este tipo de modelagem possui eficiência questionável entre alguns setores da engenharia de *software*, porém seu uso em larga escala na análise de requisitos ainda o torna relevante.

Diagrama de fluxo de dados (DFD): São diagramas que consideram a entrada, processamento e posterior saída de dados do sistema. Podem ser construídos em diferentes níveis de abstração, com o nível zero sendo o mais abstrato, representando inteiramente o sistema. Aqui os objetos são representados por setas com rótulos e suas respectivas transformações por círculos. Os elementos de entrada e saída dos objetos são representados por retângulos. A necessidade da criação de diferentes níveis de diagramas, ou seja, de diferentes diagramas mostrando maior detalhamento do processamento faz com que o DFD seja uma solução mais restrita a alguns casos específicos, em que o processamento não possua nível de complexidade tão alto.

Diagrama de fluxo de controle: Para casos em que além dos dados gerados e processados, necessita-se de um controle dos mesmos, como informações de desempenho e manutenção por exemplo, diagramas de fluxo de controle passam a ser mais apropriados. Neste caso o controle é descrito por álgebra booleana, através de mecanismos de liga/desliga ou uma lista de condições, como níveis e intertravamentos.

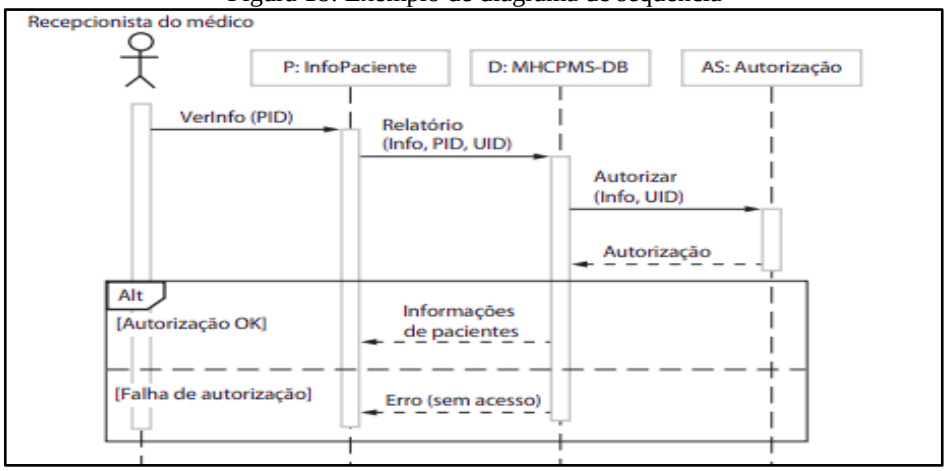
Modelagem comportamental: Se o sistema a ser construído possui um comportamento dinâmico, modelos comportamentais passam a ser apropriados pois descrevem reações do sistema de acordo com tempo ou eventos específicos. Neste caso, os casos de uso geram eventos que afetam a dinâmica do sistema, ou seja, fazem com que o sistema produza comportamentos específicos para cada ato respectivo.

Diagrama de estados: A representação desta mudança de comportamento do sistema é considerada como um grupo de estados diferente do mesmo. Desta forma, constrói-se um modelo que considera um grupo de classes e os eventos que mudam seu estado. De maneira similar aos diagramas de fluxo, os eventos que provocam mudança de estado são representados por setas rotuladas, os objetos por retângulos de pontas suaves e círculos indicando início ou fim dos eventos (como nos diagramas de atividades). Os objetos em questão podem também possuir atributos e operações.

Diagrama de sequência: Representa a transição de eventos entre os objetos em questão em função do tempo. Considerado como uma abreviação do caso de uso (PRESSMAN, 2012), representam-se neste diagrama as principais classes e o fluxo de eventos transitando entre elas. Os eventos são novamente representados por setas rotuladas, mas o diferencial deste caso é a representação de uma estimativa do tempo de processamento como um retângulo medido na vertical, ou seja, sua medida de comprimento representa o tempo para uma atividade ser concretizada. O objeto pode estar também ativo em vários momentos diferentes, assim um mesmo objeto pode possuir mais de um retângulo a ele relacionado demonstrando diferentes tempos de ativação.

A Figura 18 a seguir demonstra um diagrama de sequência como exemplo sobre as relações de uma recepcionista de um médico com o paciente e departamentos de um hospital:

Figura 18: Exemplo de diagrama de sequência



Fonte: (SOMMERVILLE, 2011).

3.2.7 Projeto e Construção

A análise de requisitos através da modelagem e anterior comunicação estabelece restrições e preceitos que serão a base da construção do *software* em si. Os desenvolvedores passam a ter maior clareza de quais direções a se tomar e então o foco se volta ao projeto.

O projeto de *software* engloba, normalmente, uma divisão entre arquitetura, componentes e interface.

A arquitetura é definida como a estrutura ou organização de componentes de programa, como se interagem e a estrutura dos dados usada por eles (PRESSMAN, 2012). Estes componentes podem variar desde os simples módulos a uma maior robustez como um banco de dados que possibilite a configuração de uma rede de servidores e clientes. Comparando-se a um esqueleto do *software*, a arquitetura é o que possibilita que diferentes soluções possam ser empregadas a partir uma solução inicial quando ocorre certa semelhança entre elas. Através do projeto de arquitetura definem-se as estruturas gerais de dados e programas do *software*.

Denominam-se componentes os blocos construtivos modulares para *software* de computador (PRESSMAN, 2012). São partes implementadas na arquitetura, é onde se incorpora a lógica de processamento de dados, as estruturas internas para implementá-la e a interface possibilitando a entrada e saída dos mesmos. Segundo Pressman (2012), o projeto de componentes está inserido em modelos da arquitetura e é onde ocorre o refinamento e detalhamento das classes relacionadas ao domínio do problema, anteriormente representadas nos modelos de requisitos, ou seja, rebaixando ainda mais o nível de abstração com relação à modelagem de requisitos.

Por fim, cita-se o projeto de interface como uma focalização da interação homem-máquina. Com o advento de mecanismos de interação conhecidos como interface gráfica do usuário (GUI – *graphical user interface*) minimizaram-se muitos dos problemas relacionados a como construir de maneira apropriada esta interação.

3.3 LINGUAGEM *PYTHON*

De acordo com o *site* oficial da linguagem de programação *Python*, a definição do que seria esta linguagem é traduzida como uma linguagem orientada a objetos. De maneira mais detalhada tem-se:

> O *Python* é uma linguagem de programação interpretada, interativa e orientada a objetos. O mesmo incorporou módulos, exceções, tipagem dinâmica, tipos de dados dinâmicos de alto nível e classes. Possui interfaces para muitas chamadas e bibliotecas do sistema, bem como para vários sistemas de janelas, e é extensível através de linguagem como o C ou C++. Também é utilizado como linguagem de extensão para

> aplicativos que precisam de uma interface programável. Finalmente, o *Python* é portátil: o mesmo pode ser executado em várias variantes do *Unix*, incluindo *Linux* e *Mac*, e no *Windows*.
>
> (VAN ROSSUM, 2001).

A linguagem possui também uma ampla gama de variedades de bibliotecas-padrão, possibilidade de se trabalhar com grande volume de dados de maneira mais simplificada e interface gráfica com diversas funcionalidades, além de fácil exibição de erros e praticidade de organização dos códigos.

Outra característica importante é que ela é uma linguagem interpretada, ou seja, não necessita de compilação visto que o interpretador lê e executa os códigos diretamente.

A principal desvantagem de *Python* com relação a outras linguagens é o seu maior tempo de processamento, justamente pelo alto nível de abstração e compactação da linguagem. Em situações onde o tempo de processamento é fundamental seu emprego pode não ser a melhor solução. Outro ponto crítico é a descentralização de informações, com vários ambientes e documentação presentes em diferentes lugares, podendo ser um empecilho para a adaptação de programadores iniciantes.

Para desenvolvimento de GUI em *Python* pode-se lançar mão de vários tipos de *frameworks*, ou *toolkits*, que consistem em implementação de um conjunto de *widgets* (elementos de interface gráfica), normalmente implementados como bibliotecas, podendo também serem implementados como uma plataforma separada ou em IDE independente. Os *frameworks* mais comuns, considerando os sistemas operacionais mais utilizados atualmente, são: *Kivy*, *PyQT*, *Tkinter*, *WxPython*, *PyGUI* e *PySide* (FATIMA, 2017). Dos principais *frameworks* em uso, o único presente em biblioteca padrão do *Python* é o *Tkinter*.

3.3.1 Interface Gráfica do Usuário com *Tkinter*

Explicitando de maneira mais rigorosa, o *kit* de ferramentas (*toolkits*) presente como parte integrante do *Python* é chamado *Tk*, cujo pacote e extensão chama-se *Tkinter*, com o nome provindo de *Tk interface*. O *Tk* foi desenvolvido incialmente para a linguagem *Tcl* (*Tool Command Language*) e consiste em uma biblioteca contendo elementos básicos de GUI podendo ser aplicado em várias linguagens, como *Ruby*, *Ada* e *Haskell*. Sua aplicação em *Python* chama-se *Tkinter*.

De acordo com a própria documentação da biblioteca a principal virtude do *Tkinter* é a presença junto à linguagem *Python* por padrão, rapidez e grande quantidade de material de referência. Para criação de interfaces mais simples em projetos de menor porte, esta biblioteca apresenta uma sensível utilidade (LUNDH, 2009).

Definições básicas em *Tkinter*:

- *Toplevel( )*: nova janela, considerado um *widget*.
- *Frame*: bloco em que será inserido um elemento, o *frame* é como uma base para a inserção de *widgets* dentro de si. Possui forma retangular.
- *Widget*: Elemento genérico de interface, podendo ser botões, caixas de seleção, caixas de texto, *frames*, entre outros.

Funções de construção: Ao se criar um *widget* ele necessita de uma função de construção, também chamada de método, possibilitando a visualização do mesmo na interface. A função mais simples é a função *pack( )* que automaticamente agrupa *widgets* dentro das janelas. Outras funções de construção a se destacar são *place( )*, que requer uma definição do ponto exato em pixels onde será colocado o *widget* e a *grid( )*, que agrupa os *widgets* em organização matricial através de linhas e colunas.

Cada uma das funções de construção possui opções de alteração detalhadas nas referências bibliográficas sobre o assunto. Observa-se que as funções são aplicadas diretamente nos *widgets* de maneira a abordá-los como objetos.

Os *widgets* também podem ter características alteradas, como cor de fundo, cor e largura da borda, efeitos em 3D como sombreamento e, obviamente, tamanho (incluindo janelas). Os botões em especial necessitam de vinculação de uma função ao serem ativados para serem apropriadamente construídos.

3.3.2 Comunicação com *Hardwares*

A linguagem *Python* propicia comunicação com hardwares externos de maneira relativamente prática. Através de poucos comandos pode-se implementar uma comunicação eficiente seja do tipo cliente-servidor, banco de dados, utilizando protocolos industriais como *Modbus* e *Profibus*, comunicação serial ou *wireless*.

Através do módulo *Pyserial* o *software* pode ter acesso a uma porta serial externa conectada ao dispositivo onde o mesmo é executado através de poucas linhas de comando.

3.4 COMUNICAÇÃO SERIAL

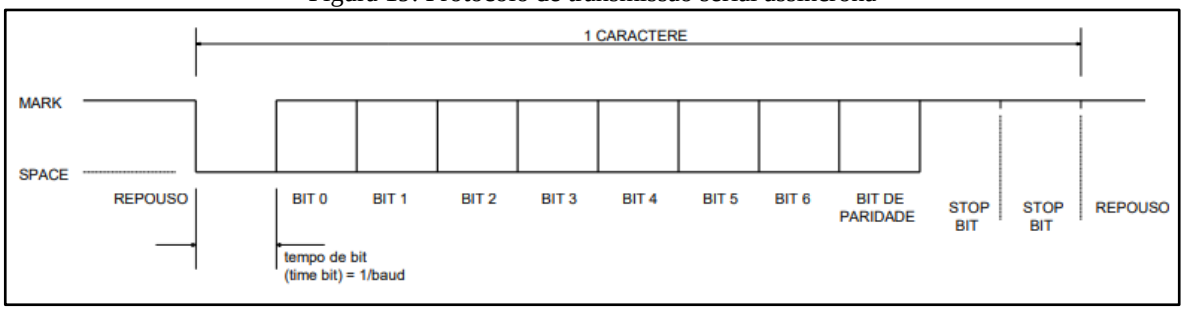
Define-se como comunicação serial a transferência de dados processada em uma única linha, de acordo com um encadeamento linear dos bits. Esta comunicação pode ser síncrona, onde os bits são pareados de acordo com um sinal de *clock*, com transmissor e receptor compartilhando a mesma frequência de *clock*. Quando os envolvidos não compartilham o

mesmo *clock* a comunicação é dita assíncrona, possuindo assim bits de controle para transmissão e recepção (LINDBLOM, 2014).

A velocidade de comunicação de maneira mais geral é dada com o termo *baud rate*. Este termo indica a quantidade de bits enviada por segundo no canal. A transmissão, representada na

Figura 19 a seguir, é controlada de acordo com os bits MARK e SPACE.

Figura 19: Protocolo de transmissão serial assíncrona

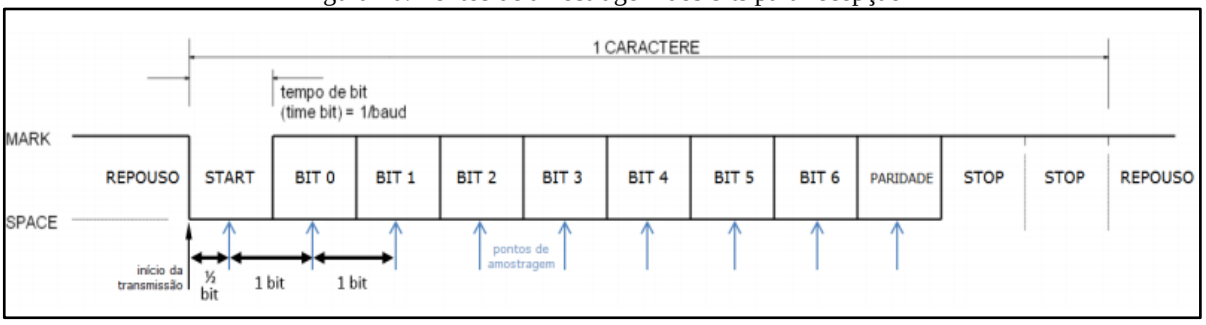


Fonte: (EPUSP, 2016)

O sinal MARK encontra-se, em estado inicial de repouso, em estado ALTO e o sinal SPACE em estado BAIXO. É a mudança de estado de SPACE que representa o início da transmissão. Transmitem-se então os bits de dados, formando um caractere em ASCII, o bit final de paridade e dois STOP BITS antes do sistema voltar ao repouso.

Como o processo de comunicação é assíncrono, o circuito de recepção amostra os bits de dados no meio de cada intervalo, com metade do *baud rate* após o início da transmissão. Na Figura 20 tem-se a demonstração dos pontos de amostragem na recepção:

Figura 20: Pontos de amostragem dos bits para recepção



Fonte: (EPUSP, 2016).

O código ASCII relacionado consiste em uma representação de um conjunto de caracteres alfanuméricos, possuindo sete bits de informação e um de paridade, através dos quais

se diferenciam números, letras em maiúsculas ou minúsculas e alguns símbolos como os de pontuação.

3.5 COMUNICAÇÃO *WIRELESS*

À transferência de informação entre dois ou mais pontos envolvidos sem utilização de cabeamento dá-se o nome de comunicação remota, mais popularmente comunicação *wireless*. As redes sem fio utilizam-se de processamento digital de sinais através de ondas de rádio moduladas, possibilitando assim a transmissão de dados. Ao longo das últimas décadas, desenvolveram-se vários tipos de redes sem fio, destacando-se:

Redes WWAN (*Wireless Wide Area Network*): São redes celulares de telefonia. Inicialmente desenvolvida para comunicação de voz, sofreram atualizações para possibilidade de transmissão de dados.

Redes WMAN (*Wireless Metropolitan Area Network*): São redes de longo alcance que abrangem diversas redes locais e estabelecem conexão entre elas.

Redes WLAN (*Wireless Local Network*): São redes que possibilitam a interconexão de dispositivos numa área de curto alcance. Os equipamentos são conectados a um servidor local que se comunica via satélite com uma rede de Internet.

Redes WPAN (*Wireless Personal Area Network*): São redes de curtíssima distância entre dispositivos que se estabelecem em uma comunicação *ad-hoc*, ou seja, onde apenas estes dispositivos isolados podem estar conectados. Um exemplo deste tipo de rede que se popularizou nos últimos anos é a tecnologia *Bluetooth*.

4 MATERIAIS E MÉTODOS

4.1 ESTUDOS TEÓRICOS

Para os estudos teóricos de Engenharia de *Software* foram utilizadas as bibliografias citadas nas referências, destacando-se as referências (PRESSMAN, 2012) e (SOMMERVILLE, 2011) e para UML, a referência (RUMBAUGH, BOOCH, JACOBSON, 2006).

A partir dos estudos teóricos, elaborou-se uma metodologia de desenvolvimento do *software* baseada em preceitos de engenharia de *software*.

4.2 ANÁLISE DE REQUISITOS E MODELAGEM

Através da atividade metodológica de comunicação, com *e-mails* e trocas de mensagens com a orientadora, definiram-se de maneira geral quais os requisitos para a produção do *software* em questão, que tem como objetivo principal o estabelecimento de um controle de um dispositivo robótico externo como instrumento de TA, posteriormente a ser empregado pelo Grupo de Estudo de Tecnologia e Acessibilidade. Realizou-se também um planejamento de cronograma inicial para as próximas atividades através do plano de projeto.

Elaboraram-se então modelos de requisitos para maior clareza sobre tarefas e serviços a serem entregues pelo *software*. Para esta modelagem empregou-se a sintaxe de UML, tendo como resultados os diagramas de caso de uso e de classe.

4.2.1 Requisitos de Acessibilidade

- **Varredura Automática:** A varredura automática é um dos principais requisitos de acessibilidade para a implementação do *software*. Ela deve controlar todas as funcionalidades do *software* e o acionamento deverá ser realizado por qualquer dispositivo acionador que possa substituir as funções do botão esquerdo do *mouse*. Uma vez iniciado o *software*, o usuário deve ser capaz de executar as funções sem a ajuda de terceiros.
- **Interface:** A interface deve ser simples, lúdica e conter o menor número de opções possíveis, para que a varredura seja efetiva.

4.3 IMPLEMENTAÇÃO COMPUTACIONAL

Após a modelagem de requisitos, iniciou-se o processo de desenvolvimento do *software* e, para isso, com base na linguagem de programação Python, e na sua biblioteca de GUI relacionada (*Tkinter*) iniciou-se o procedimento de codificação.

A codificação foi realizada no ambiente de desenvolvimento integrado *Pycharm Community Edition*. Realizou-se um estudo sobre comunicação com hardwares, incialmente projetou-se com comunicação serial através de cabeamento USB em placa de prototipagem eletrônica (tomou-se a placa *Arduino R3* como exemplo) para controle de motores do dispositivo robótico automóvel e por fim se elaborou também um projeto de comunicação *wireless* via rede WLAN com uma placa de microcomputador *Raspberry Pi* como exemplo.

4.4 DESIGN DE INTERFACES

Por fim para construção do *design* voltado ao público infantil utilizou-se de ferramentas de desenho como *MS Paint 3D* e *Adobe Photoshop CC 2015*, sendo o primeiro para o *design* do menu principal e o segundo para a edição do *design* dos botões também presentes no menu principal.

4.5 CONSTRUÇÃO DO PROTÓTIPO E TESTES

Devido à ausência de aulas presenciais relacionada a pandemia causada pelo Novo Coronavírus neste ano, não foi possível a construção do kit robótico no LTAD. Com isso este trabalho apresenta o resultado final do *software* como um protótipo, mostrando também projetos de implementação e alterações necessárias para sua aplicação na prática.

Embora seja apresentado como um protótipo, salienta-se que o programa se encontra muito próximo de sua versão final, com os ajustes necessários para sua atualização sendo explicitamente abordados neste trabalho.

5 DISCUSSÃO E RESULTADOS

O desenvolvimento do *software* em questão foi realizado obedecendo a metodologia empregada comumente em engenharia de *software*. Inicialmente levantaram-se os requisitos mais gerais e objetivos principais, através de mensagens por redes sociais e *e-mail*. Planejou-se como se daria a realização do processo, escolhendo-se o modelo espiral, devido à natureza mais clara dos requisitos iniciais e à sua vasta aplicação na prática e não exige experiência em desenvolvimento de projetos, como os métodos ágeis.

5.1 MODELAGEM DE REQUISITOS

A modelagem de requisitos foi realizada com o intuito de trazer maior clareza para o desenvolvedor acerca dos requerimentos de serviços e tarefas a serem disponibilizados.

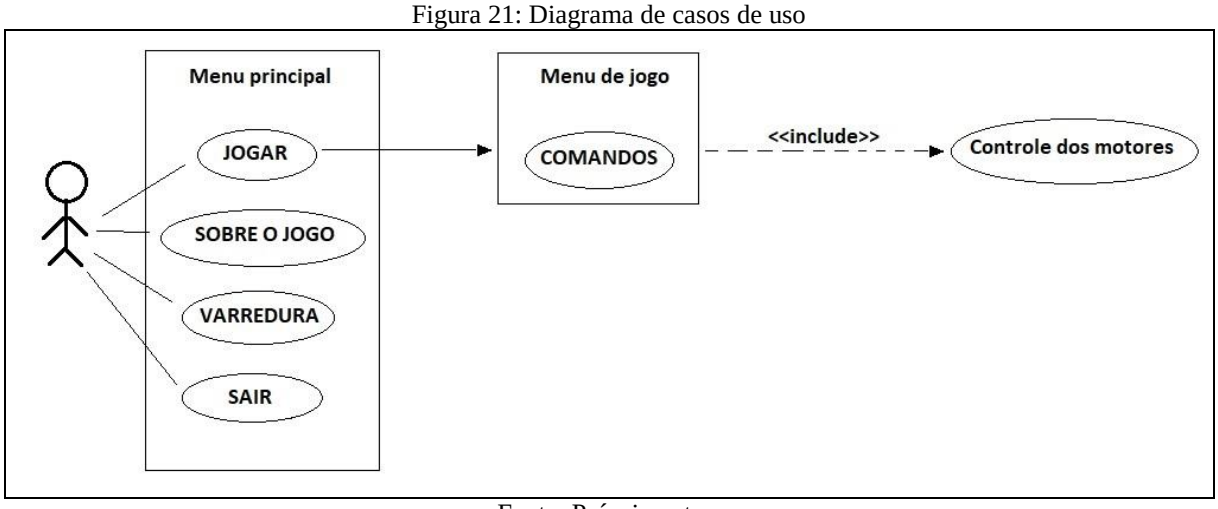
Tendo como principal objetivo o controle de um pequeno automóvel à distância, a quantidade de dados fluindo pelo *software* não chega a ser um valor crítico. Como o usuário não realizará mais de uma atividade simultaneamente, uma análise temporal também não se faz necessária, pois não terá muita influência.

5.1.1 Diagrama de Casos de Uso e Diagramas de Classes

Com estas considerações, tem-se análise sobre possíveis casos de uso pelo sistema, assim como o estabelecimento de classes, seus atributos, operações e possíveis relações entre si.

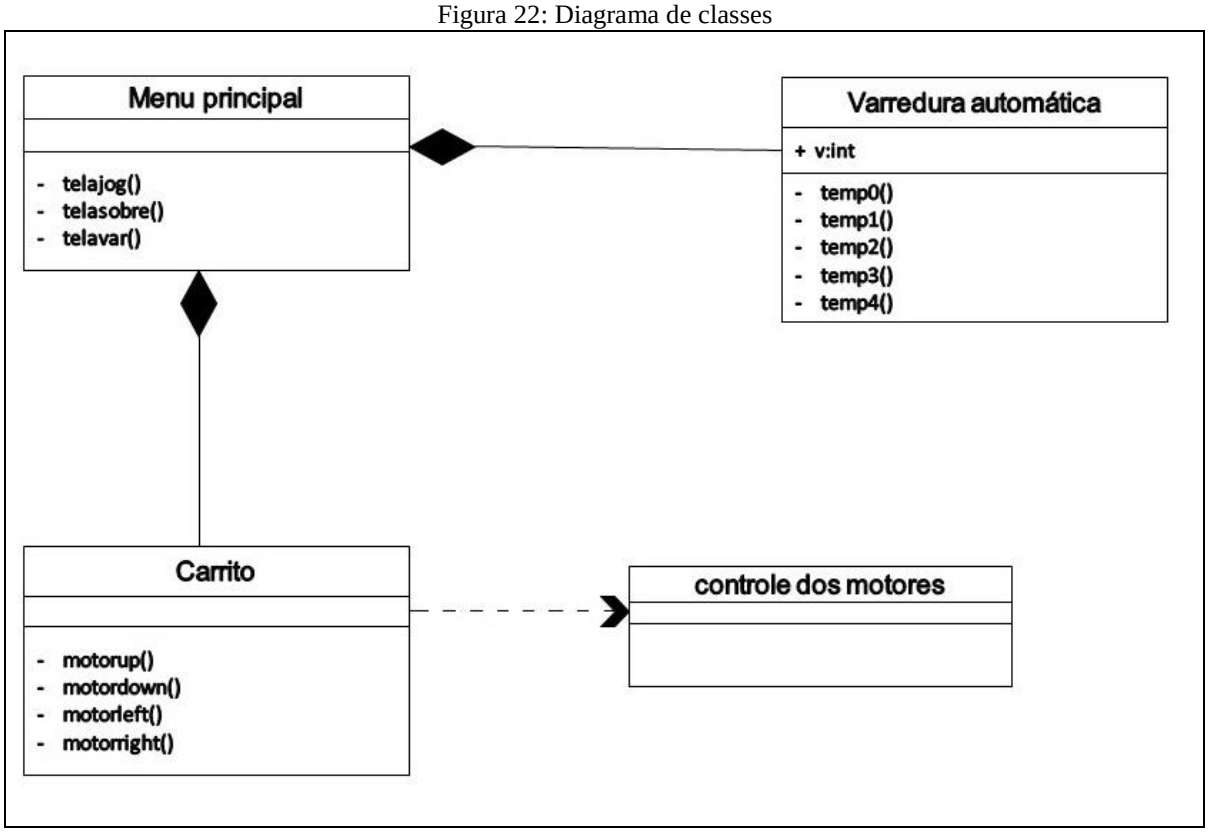
Elaboraram-se então, diagramas de caso de uso e de classes representados, respectivamente, na Figura 21 e na Figura 22:

Figura 21: Diagrama de casos de uso



Fonte: Próprio autor.

Figura 22: Diagrama de classes

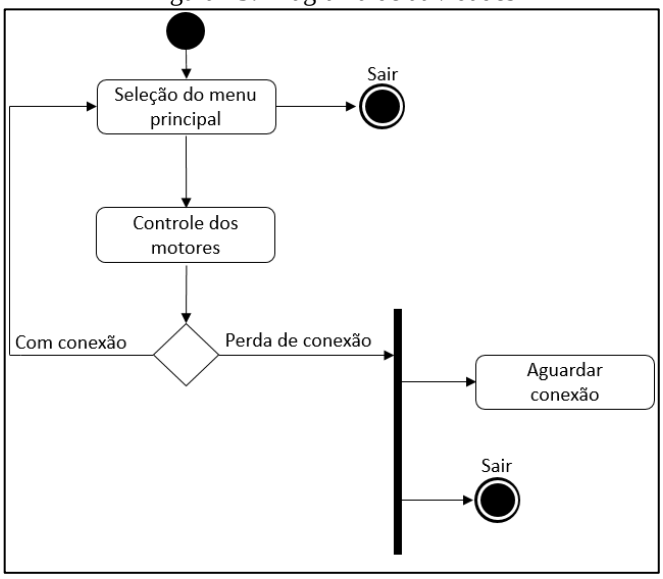


Fonte: Próprio autor.

5.1.2 Diagrama de Atividades

Construiu-se também um diagrama de atividades para demonstração do que ocorre no menu de controle quando há perda de conexão entre o computador que executa o *software* e o carrinho, presente na Figura 23 a seguir.

Figura 23: Diagrama de atividades



Fonte: Próprio autor.

Com a perda de conexão devido a um evento externo, podendo ser devido a perda de conectividade *wireless* por perda de alcance ou, no caso de se utilizar comunicação serial, a retirada do cabo, entre outras anomalias, sugere-se a criação de uma interface de alerta com a opção de aguardar a volta da conexão ou a saída direta do programa, tendo este diagrama a finalidade de prever um tratamento de erros e um protocolo para segurança para estes casos.

A demonstração parcial feita apenas com o menu de controle foi realizada pois as demais interfaces não produzem atividades que provocam alterações consideráveis no sistema, sendo apenas interfaces em que o usuário abre e depois pode apenas voltar ao menu principal.

Os diagramas foram montados através do *software MS PowerPoint*. No diagrama de casos de uso ressalta-se que a ação de controle de motores só pode ser executada enquanto o menu de jogo, explicitado no diagrama de classes com o título "Carrito", estiver ativo.

O menu principal executa as operações telajog(), telasobre() e telavar() acionando as janelas, respectivamente relacionadas ao menu de controle do carro, o menu de informações e menu de varredura automática. Existe ainda uma configuração para encerramento do aplicativo associada a uma função própria do *Tkinter*, por isso não explicitada no diagrama, com isto se repetindo para os menus derivados.

A varredura automática ocorre através do atributo "v" exercendo um papel de variável global, cuja atuação incorre sobre todo o sistema. As operações explicitadas na classe "Varredura automática" se relacionam aos diferentes estados de comando dos botões de acordo com o valor de "v", sendo de 1s, 1,25s, 1,5s, 1,75s e 2s.

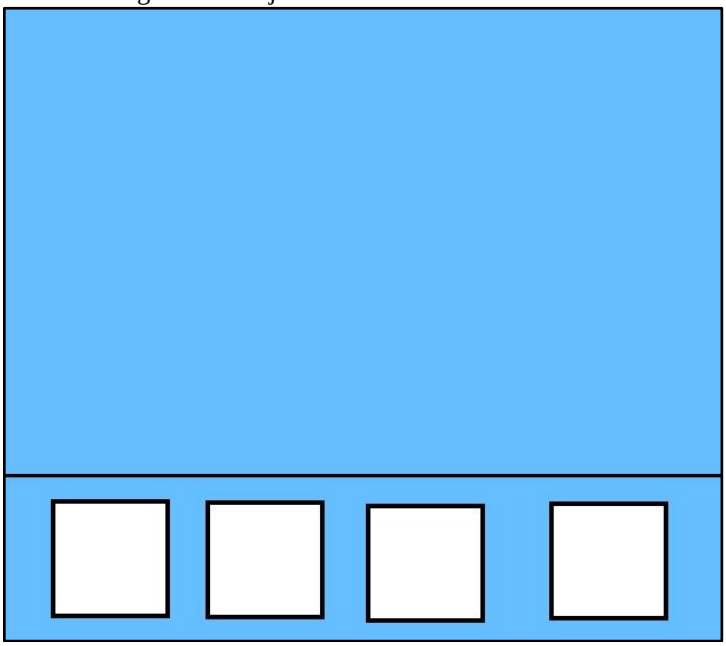
Por fim, na classe de construção do menu "Carrito" as operações se relacionam ao controle dos motores, podendo ser um comando que cause um movimento para frente, para trás ou para os lados.

5.2 IMPLEMENTAÇÃO COMPUTACIONAL

Utilizou-se para a construção do *software* a IDE *Pycharm Community Edition 2019.1.1*, que possibilita uma organização prática da codificação, com opções como minimização de classes e aplicações, alertas de sintaxe e sugestões de correção.

A interface do menu principal foi projetada de acordo com a exibição dos quatro elementos de comando em conjunto a um *design* acessível para crianças. O projeto de interface do menu principal, realizado através de *software MS Paint 3D* e é apresentado na Figura 24.

Figura 24: Projeto de interface do menu inicial



Fonte: Próprio Autor.

O espaço acima dos quatro botões contempla uma imagem relacionada ao tema do *software*, de acordo com o *design* mencionado anteriormente.

A interface então foi criada através de funções relacionadas à biblioteca Tkinter, como novas janelas (*Toplevel( )*), botões (*Button( )*), elementos textuais (*Label( )*) e de imagem (*Canvas( )*).

5.2.1 Criação da Varredura Automática

Um dos requisitos principais de TA a ser implementado é a varredura automática, constituindo-se de uma seleção automática varrendo pelos *widgets* presentes nas interfaces, representada por uma borda de cor vermelha indicando esta seleção.

Este recurso foi reproduzido com o auxílio do método *after( )*, que possibilita a execução de outras funções após decorrido um período tempo. Um exemplo da aplicação do método, presente na codificação do próprio *software* referido encontra-se na Figura 25.

Figura 25: Código com exemplo de aplicação do método after()

```
def var1(self):
    self.widget1["bd"] = 4
    self.widget2["bd"] = 0
    self.widget3["bd"] = 0
    self.widget4["bd"] = 0

    #Setando tecla Enter como meio de comando
    self.sair.focus_set()
    root.bind("<Return>", lambda e: quit())

    self.widget1.place(x=475, y=hbut)
    self.widget2.place(x=25,y=hbut)
    self.widget3.place(x=175, y=hbut)
    self.widget4.place(x=325, y=hbut)
    self.widget4.after(v, self.var2)
```

Fonte: Próprio Autor.

Observa-se que o método é aplicado a um objeto, no caso um *widget*, ou seja, é aplicado no momento da construção e exibição do mesmo. Os parâmetros definidos no caso foram, respectivamente, o tempo de espera para sua aplicação (denotado pela variável "v") e a função a ser executada após este ínterim.

A variável "v", explicitada na modelagem de requisitos, é necessária para a que seja possibilitada a mudança de tempo de varredura, podendo ser de valor 1s, 1,25s, 1,5s, 1,75s ou 2s. É ela quem indica o tempo atual de varredura. Para a correta indicação de seleção do botão correspondente utilizou-se o método *focus_set( )* que força o foco sobre o objeto em que é aplicado e também o método *bind* que assimila uma tecla do teclado a um comando. No caso do exemplo em questão o comando gera a saída do aplicativo.

Um dos primeiros problemas encontrado na produção da varredura automática foi a representação da seleção de cada botão. Por algum problema de compatibilidade com o sistema operacional *Windows* os *widgets* de botão não apresentam mudança de atributo de cor de borda. A solução encontrada para "simular uma borda" consistiu em configurar a cor de fundo dos *frames* em que os botões estavam inseridos para então mudar o valor da largura da borda dos *frames*, sem alterar configurações dos *widgets*. Este atributo de largura da borda é delimitado pela sigla "bd" de *borderwidth* cujo valor em 0 indica que o *frame* não possui borda excedente, e valores acima de zero indicam o tamanho da borda em excesso.

Com isso, cada operação de *temp( )* presente no diagrama de classes, pertencente à classe "Varredura Automática" realiza esta mudança de atributos dos *frames* junto ao método de construção, no caso o *place( )*, para a correta exibição destas mudanças. Estas operações se repetem nas outras classes, aplicando-se a varredura automática em todo o sistema, porém são apenas consequência das operações realizadas no menu de varredura, pois utilizam a variável

global "v" sem alterá-la. Um exemplo do que ocorre no menu de varredura encontra-se na operação *temp1( )*, utilizada como comando dos botões, representado na Figura 26:

Figura 26: Código da função temp1( )

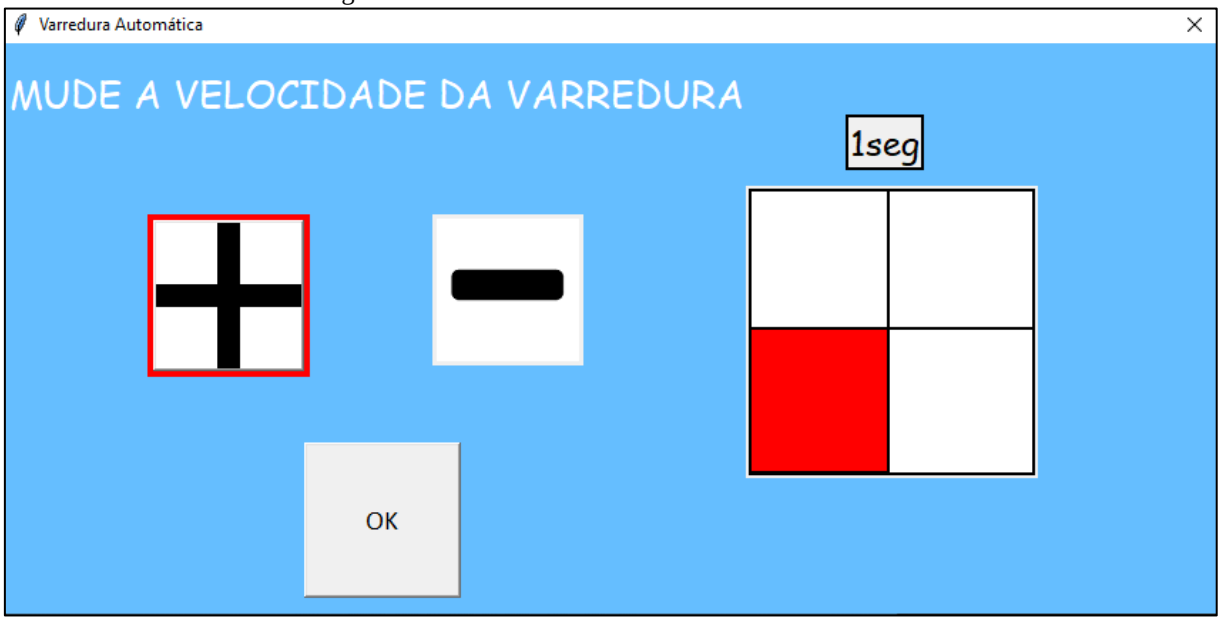
```
def temp1(self):
    global v
    if v==1250:
        self.temp2()
    if v==1500:
        pass
    if v==1750:
        self.temp4()
    if v==2000:
        self.nada()

    v = 1250
    variable.set("1,25 seg")

    self.texto.pack()
    self.mas["command"] = self.temp2
    self.menos["command"]=self.temp0
```

Fonte: Próprio Autor.

Nesta função a variável "v" muda para 1250. Como o método *after( )* possui parâmetro em milissegundos, então a variável tem como atribuição o valor 1,25s. Esta função é chamada quando o botão "mais" é acionado e "v" está com valor 1s ou o botão "menos" é acionado com "v" de valor 1,5s. O texto exibido no menu indicando o valor atual de tempo de varredura também é uma variável e sua exibição muda de acordo com o valor de "v". Por fim, os comandos relacionados aos botões também devem mudar, indicando um valor de tempo de varredura mais rápido (*temp0* que setará v=1s) ou mais lento (temp2 que setará v=1,5s). As outras quatro operações de *temp( )* funcionam com a mesma lógica, conforme os possíveis valores para "v".

A Figura 27 demonstra a interface do menu de varredura automática. Observa-se a presença de objetos geométricos construídos na lateral a partir da função *Canvas( )* como indicação da varredura.

Figura 27: Interface do menu de varredura automática



Fonte: Próprio Autor.

Alguns recursos foram disponibilizados com base em atributos relacionados às janelas, como a não-permissão do usuário para alterar o tamanho da mesma e o bloqueio de utilização do menu principal enquanto os menus derivados estiverem ativos. O primeiro foi conseguido através do atributo *resizable(0,0)* enquanto que o último foi realizado com o uso dos métodos *transient( )* e *grab_set( )* aplicados junto ao objeto relacionado a cada janela derivada, como indicado a Figura 28.

Figura 28: Métodos transient( ) e grab_set( )

```
class Application2:


    def __init__(self,master2=None):

        new = Toplevel(bg=bgd)
        new.title("Varredura Automática")
        new.geometry("850x400")


        # Impedir o usuário de utilizar a janela principal
        #enquanto outra janela está aberta
        new.transient(root)
        new.grab_set()
        new.resizable(0,0)
```

Fonte: Próprio Autor.

Estas funcionalidades foram implementadas em todas as janelas derivadas do menu raiz.

5.2.2 Interface do menu principal

O menu principal foi criado de acordo com o projeto representado pela

. O botão "Jogar" constrói a janela que permitirá o controle do carro, o botão "Sobre" constrói a janela que mostra mais informações sobre o *software*, enquanto que "Varredura" leva ao menu de varredura automática mostrado na **Erro! Fonte de referência não encontrada.** 27. P or fim, o botão "Sair" fecha a aplicação.

A representação da interface do menu principal encontra-se presente na Figura 29. A imagem acessível ao público infantil foi editada através do *software MS Paint 3D*, enquanto que o *design* dos botões foi construído a partir do *software Adobe Photoshop CC 2015*.

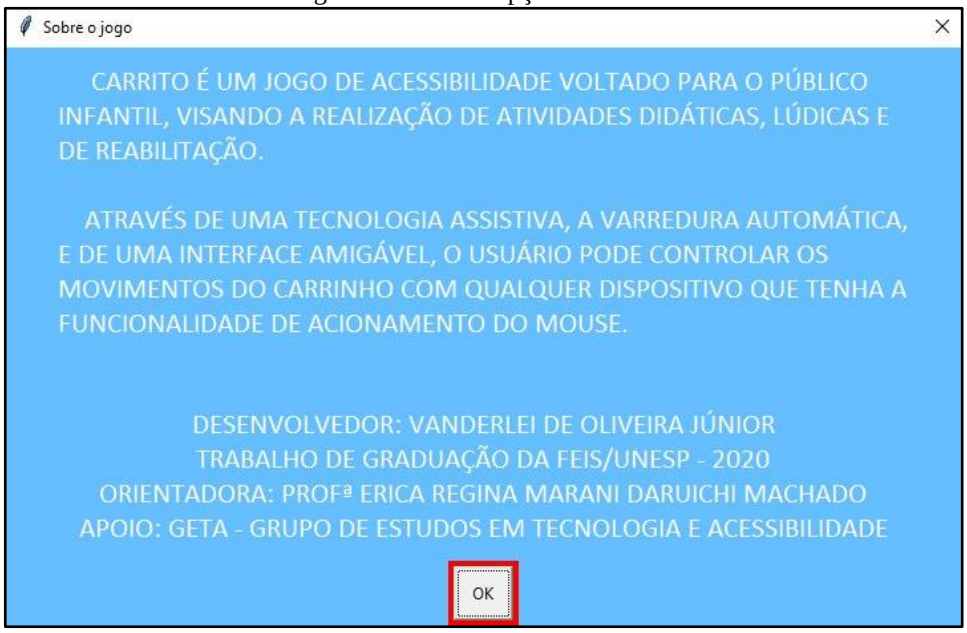
Figura 29: Interface do menu principal

Fonte: próprio autor.

5.2.3 Interface do botão SOBRE

Para esta opção foi implementada uma tela simples contendo informações sobre o jogo e um botão de retorno ao menu principal já selecionado, conforme mostrado na Figura 30 a seguir.

Figura 30: Tela da opção "SOBRE"

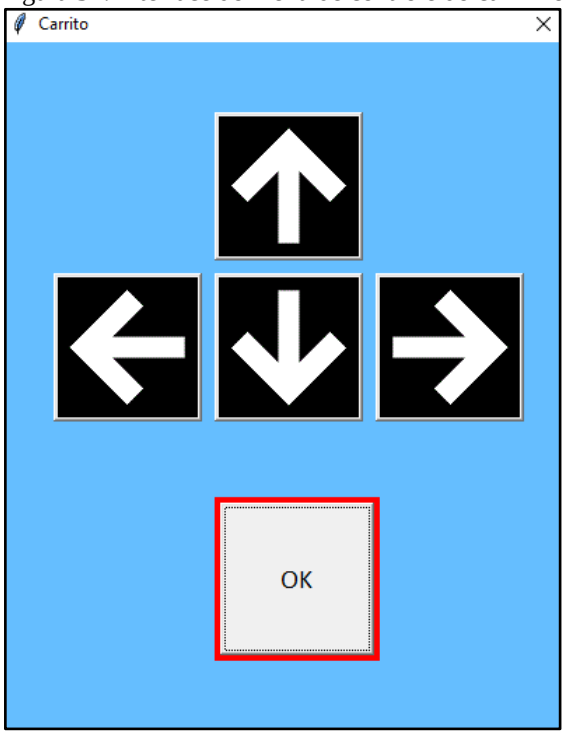


Fonte: próprio autor.

5.3 INTERFACE DO MENU DE CONTROLE E COMUNICAÇÃO

Apresenta-se finalmente a interface construída para o menu de controle do carrinho, representada na Figura 31. As opções de controle são: movimento para trás, movimento para os lados e ré, representadas pelos quatro botões de comando a elas relacionados. Ademais, um botão de retorno também foi estabelecido.

Figura 31: Interface do menu de controle do carrinho



Fonte: próprio autor.

5.4 CONFIGURAÇÃO DO *HARDWARE*

Para o controle dos motores do carro realizou-se um estudo sobre os tipos de comunicação mais adequados com hardwares externos e, inicialmente, testou a comunicação serial através de cabo USB com uma placa de prototipagem eletrônica *Arduino Uno R3*.

Comunicação serial: Uma forma muito simplificada de comunicação, necessitando de poucas linhas de código em *Python* assim como em *Arduino* para controle direto dos motores. Através da biblioteca *pyserial* se estabelece uma conexão com uma porta USB, instaurando-se um barramento de dados (LIECHTI, 2001). Pode-se então executar funções de escrita e leitura de dados presentes neste barramento. Como exemplificado na Seção 2.4, a transmissão é realizada em bytes codificados em ASCII. O código é apresentado na Figura 32.

Figura 32: Classe Application3

```
class Application3:

    def __init__(self, master3=None):
        self.ser = serial.Serial('COM3', baudrate=9600, timeout=1)
```

Fonte: Próprio Autor.

A função Serial( ) apresenta como atributos a porta USB disponível, a velocidade de transmissão dos dados em bits por segundo (baudrate) e por fim o tempo de espera para leitura de dados no barramento em segundos (timeout).

Um dos primeiros problemas que surge com deste tipo de comunicação é o fato de se dever explicitar em qual porta USB está conectado o hardware em questão. Uma automatização desta verificação pode ser feita através de um levantamento de uma lista contendo todas as portas disponíveis no sistema através da função serial.tools.list_ports.comports( ) em conjunto com uma comparação em string com o nome do hardware a ser utilizado, como "*Arduino*" por exemplo.

Após a abertura da porta através da função *write*( ) é transferida uma informação para o barramento que será lida posteriormente pelo algoritmo em *Arduino* e, imediatamente após isto se fecha a porta, para possibilitar a leitura. Existe ainda uma verificação para o caso em que a porta se encontrar fechada, abrindo-se brevemente para a escrita do dado, veja o código da Figura 33.

Figura 33: Função Motorup()

```
def motorup(self):
    if self.ser.is_open == False:
            self.ser.open()
    else:
        pass
    self.ser.write(b'0')

    self.ser.close()
```

Fonte: Próprio Autor.

Os dados a serem transmitidos são 00, 01, 10 e 11, respectivamente lidos em decimal no *Arduino* para 0, 1, 2 e 3, precedidos pela letra "b" indicando o tipo do dano transmitido, byte (procedimento requerido em *Python 3*). De acordo com cada valor diferente presente no barramento, o segundo código em *Arduino* executa uma função referente ao controle de motores para frente, para trás ou para os lados.

Como exemplo, o código Figura 34 transmite o comando para a direção frontal, que após a leitura do valor de controle 0 no barramento executa função *cima*(), descrita na Figura 34 a seguir.

Figura 34: Função cima( )

```
void cima(int a)
{
  analogWrite(E1, a);
  digitalWrite(M1, LOW);
  analogWrite(E2, a);
  digitalWrite(M2, LOW);

  delay (1000);  // Procedimento de parada
  analogWrite(E1, 0);
  analogWrite(E2, 0);

}
```

Fonte: Próprio Autor.

O código completo encontra-se em anexo neste trabalho. Os valores de E1 e E2 referem-se à velocidade dos motores traseiros em valor absoluto enquanto que M1 e M2 referem-se à rotação em sentido horário dos motores frontais de direção em valor booleano. Estas quatro variáveis estão atreladas a pinos I/O de uso geral presentes na placa do *Arduino*, podendo transmitir um sinal analógico (para E1 e E2) ou digital (para M1 e M2) e, assim, estes pinos estarão conectados às respectivas entradas dos motores.

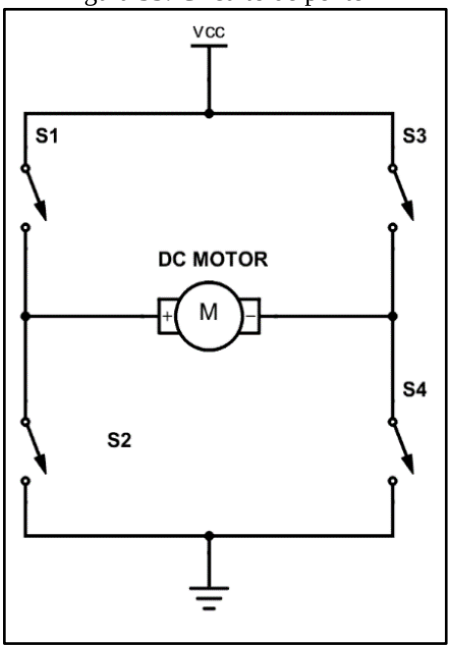
O comando de avanço frontal é o exemplificado pelo código apresentado na Figura 34, o comando de reversão é similar, apenas com mudança de M1 e M2 que passam a ter valor

ALTO, de maneira que para virar para os lados basta deixar os valores E1 e M1 positivos e E2 e M2 nulos possibilitando o giro para um lado e E1 e M1 sendo nulos enquanto E2 e M2 positivos possibilitando o giro para a outra direção. A variável "a" define a velocidade de rotação dos motores, podendo estar na faixa de 0 a 255.

Ressalta-se aqui a necessidade da conexão de uma ponte H (representada pelo módulo L298n mencionado na seção anterior) entre o *Arduino* e os motores traseiros, possibilitando o movimento de reversão.

O funcionamento da ponte H, representado na Figura 35 a seguir, ocorre da seguinte maneira: Quando S1 e S4 estão fechadas e S2 e S3 abertas, o motor encontrará uma diferença de potencial elétrico com sentido saindo da esquerda para a direita, ou seja, o lado esquerdo do motor estará fortemente polarizado enquanto que o lado direito estará ligado à terra e, por isso, com fraca polarização. Já quando S2 e S3 estão fechadas e S1 e S4 abertas o processo inverso ocorre e a corrente elétrica fluirá na direção contrária, fazendo com que o motor gire na direção reversa. Em geral os controles das chaves são feitos em pares, para S1 com S4 e S2 com S3.

Figura 35: Circuito de ponte H

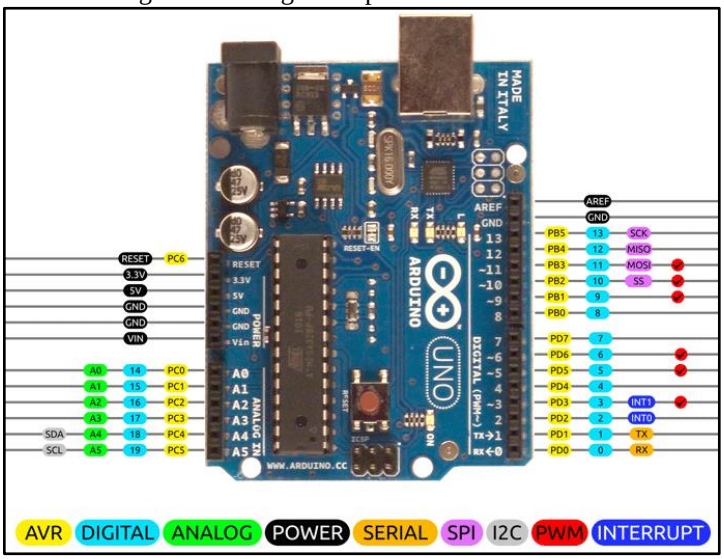


Fonte: Próprio Autor.

As chaves S1-S4 podem ser transistores bipolares de junção, sendo do tipo PNP para S1 e S3 e NPN para S2 e S4 que possuem uma entrada para controle de chaveamento.

A pinagem da placa do *Arduino Uno* considerada encontra-se presente na Figura 36. Observa-se que os pinos 14 a 19 são as únicas entradas I/O que transmitem sinais analógicos e digitais.

Figura 36: Pinagem da placa Arduino Uno R3



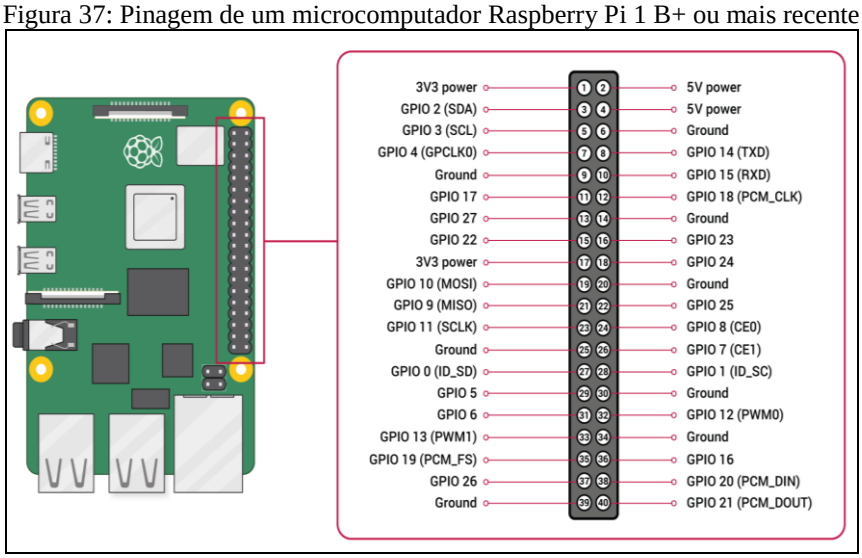
Fonte: (BOUNI, foto por Arduino.cc, 2018)

Um dos maiores problemas deste método é que, além da presença de cabos para comunicação, há a necessidade de se especificar qual porta está sendo usada e disponível para isto. Mas apesar desta desvantagem a possibilidade de sucesso na comunicação é relativamente alta, pela simplicidade da lógica de controle e não-necessidade de implementação de novos componentes eletrônicos.

Comunicação *wireless*: é um tipo de comunicação com maior complexidade, porém com melhor eficiência e praticidade. Estudou-se duas possibilidades de comunicação *wireless*, via rede WPAN através do *Bluetooth* e via rede WLAN utilizando um receptor de *Wifi*. A primeira opção tem como solução mais facilitada o controle realizado através de um módulo RS232 HC-05 conectado à placa do *Arduino*, mas com o projeto de controle sendo realizado via dispositivo *mobile* com sistema operacional *Android* (THOMSEN, 2015), fugindo do objetivo proposto que é a criação de um *software* a ser utilizado em *desktop*.

O estudo de comunicação via rede WLAN produziu como solução a utilização de outro hardware, o microcomputador *Raspberry Pi*. Este microcomputador possui grande compatibilidade com a linguagem *Python* através da biblioteca *RPi.GPIO*, que possibilita o controle dos pinos I/O de uso geral diretamente. Assim como o *Arduino*, o *Raspberry Pi* também conta com IDE própria, a *Raspian*, necessária para compilação e *upload* dos códigos para o hardware. A *Raspian* possibilita a importação e execução de códigos em *Python* com o *upload* sendo feito diretamente no hardware. A pinagem do microcomputador encontra-se presente na Figura 37, a partir do modelo Pi 1 B+ lançado em 2014 ou mais recente (os modelos anteriores possuem 26 pinos):

Figura 37: Pinagem de um microcomputador Raspberry Pi 1 B+ ou mais recente

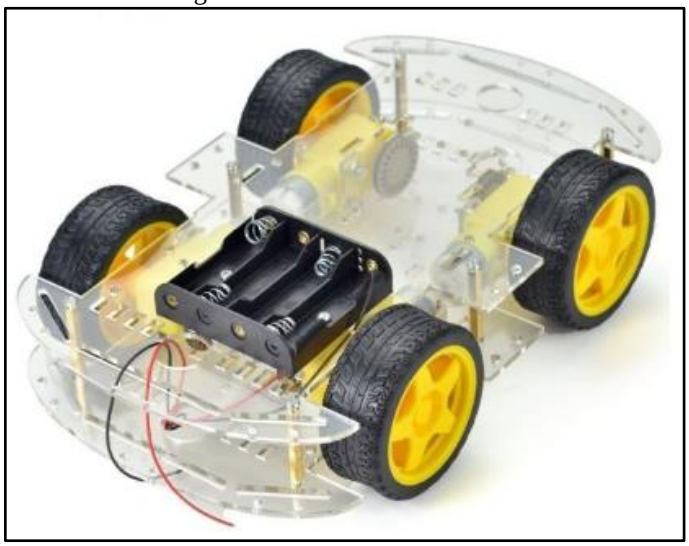


Fonte: (RASPBERRY PI FOUNDATION, 2018).

Estabelecimento de conexão WLAN: A conectividade do hardware é estabelecida através de um procedimento prévio, podendo-se utilizar de um teclado e monitor ou via cabo Ethernet. Este procedimento pode configurar o *Raspberry Pi* como um dispositivo de *access point* de uma rede WLAN funcional, servindo assim como ponto de conexão para outros dispositivos, no caso o computador a ser executado o *software* (MONDAL, 2017). Esta conexão normalmente é feita sob protocolo SSH e pode ser realizada através de *software*s específicos para este fim como, por exemplo, o *Putty*, (THE PI, 2017), (RASPBERRY PI FOUNDATION, 2018). Uma vez estando o minicomputador conectado à rede *Wifi*, é necessário também o *download* e a instalação da biblioteca *RPi.GPIO*, podendo ser feito através do *pip* (para sistema operacional Windows) ou *sudo* (Linux). Sendo assim é possível controlar o minicomputador remotamente.

Projeto de montagem do carro: Projetou-se o circuito em *protoboard* da montagem do carro e ligações dos cabos com o *Raspberry Pi*, de acordo com a pinagem do minicomputador representada na Figura 37. Assim como no projeto de comunicação serial, utilizam-se neste caso quatro pinos de uso geral para o controle dos quatro motores e a ponte H para habilitar o uso do giro em sentido reverso dos motores no movimento de ré (ARANACORP, 2018). Os pinos 7 e 15 foram selecionados para controle dos motores traseiros, já pinos 11 e 13 foram escolhidos para a rotação dos motores frontais, respectivamente, da esquerda e da direita. Um kit robótico com chassi e quatro motores DC é apresentado na Figura 38 a seguir.

Figura 38: Kit robótico montado



Fonte: (CURTO CIRCUITO, 2020)

Ressalta-se que o modelo de kit pelo qual se projeta o *software* é o mais simples sem eixo móvel frontal, sendo assim o giro para as direções laterais é feito com a movimentação dos dois motores do lado contrário à direção que se pretende girar, mantendo os outros dois desabilitados, esta configuração foi considerada para os dois casos projetados.

De maneira similar ao realizado em comunicação serial, utilizou-se quatro variáveis associadas aos quatro pinos GPIO do minicomputador. Uma diferença fundamental aqui é que os sinais de dados são transferidos diretamente para os motores, ao invés do estabelecimento de um barramento entre o computador e o *Arduino* como feito anteriormente. Com isso não há necessidade de leitura dos mesmos pelo hardware externo ou de escrita por parte do computador que executa o *software*, visto que o código do programa é executado diretamente pelo *Raspberry Pi* e apenas acessado pelo computador. Ressalta-se novamente que a escolha do *Tkinter* como criador de GUI mostra-se interessante pela compatibilidade e portabilidade com o minicomputador. Na Figura 39 mostra-se o mesmo comando do *software* indicando movimentação do carro para frente, utilizando a nova configuração com o *Raspberry Pi*:

Figura 39: Nova configuração da função motorup( )

```
def motorup(self):
    init()
    gpio.output(7, False)
    gpio.output(11, True)
    gpio.output(13, True)
    gpio.output(15, False)
    time.sleep(1)
    gpio.cleanup()
```

Fonte: Próprio Autor.

Observa-se a utilização de todas as saídas como saídas digitais desta vez. O estabelecimento dos valores dos pinos GPIO é feito pela função *gpio.output()*, enquanto que a função *time.sleep(1)* paralisa a varredura dos códigos por um segundo, com isso o carro se move neste ínterim. Por fim *gpio.cleanup()* refresca as saídas, desfazendo a configuração feita por *gpio.output()* e *gpio.setup()* e provocando a parada dos motores.

A função *init*() mostrada na Figura 40 a seguir tem como objetivo o estabelecimento dos pinos 7, 11, 13 e 15 como pinos de saída de dados:

Figura 40: Função init( )

```
def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(7, gpio.OUT)
    gpio.setup(11, gpio.OUT)
    gpio.setup(13, gpio.OUT)
    gpio.setup(15, gpio.OUT)
```

Fonte: Próprio Autor.

Então cada operação de comando relacionada aos botões do menu "Jogar" executa este novo conjunto de instruções que na prática realiza as mesmas funções que as demonstradas no projeto em *Arduino*. Para movimento de ré as saídas 7 e 15 passam para o estado *True* equivalente ao estado ALTO no *Arduino*, e as saídas 11 e 13 passam para o estado *False* equivalente a BAIXO no *Arduino*. Estas últimas quando em estados diferentes entre si fazem o carro girar para os lados, também de acordo com o controle realizado no projeto de comunicação serial.

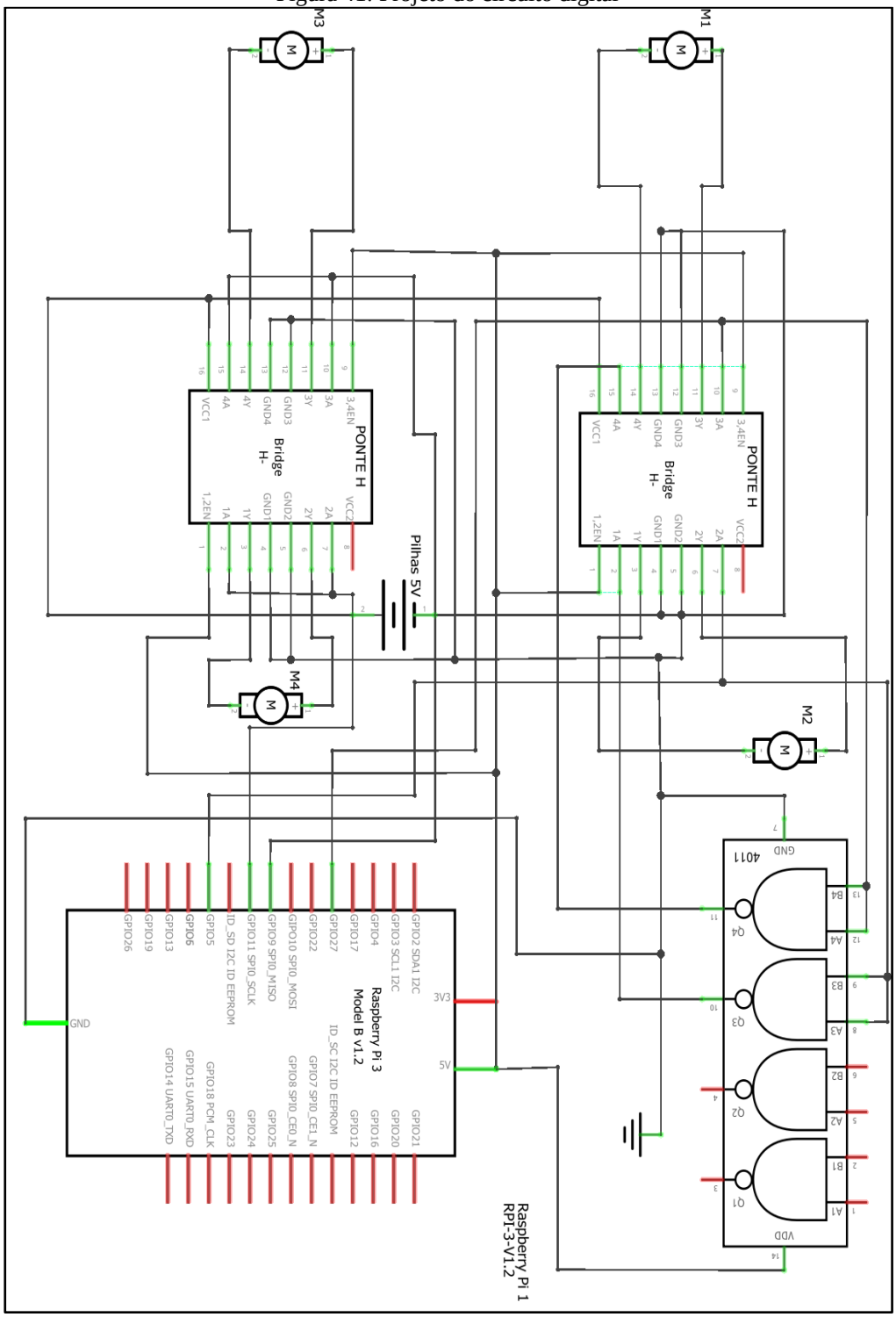
As funções *gpio.setmode()* e *gpio.setup()* definem os pinos I/O citados como saída (CROSTON, 2016). Essa definição apesar de parecer uma redundância pode ser necessária visto que no fim das operações de comando esta configuração é desfeita com a função *gpio.cleanup()*.

Projetaram-se então o circuito físico e o digital, contendo as ligações com a ponte H que irão para os motores, utilizando-se o *software Fritzing* para construção dos mesmos. Para a ponte H considerou-se o CI de "meia ponte" SN754410NE sendo o único presente no *Fritzing* para este fim, porém a mesma função pode ser realizada pelo módulo L298n considerado mais popular e mais indicado para este fim.

O CI 4011 está presente possibilitando a inversão do sinal que chega nas entradas da ponte H, pois como os pinos 7 e 15 são responsáveis pelo controle, ambos possibilitam a movimentação frontal (entradas de controle 3A e 2A respectivamente) e reversa (entradas de controle 4A e 1A. Com isto, através do auxílio do 4011 que é um CI contendo quatro portas

NAND, neste caso com entradas curto-circuitadas emulando portas inversoras, podem-se inverter estes sinais que irão posteriormente serem conectados às entradas 4A e 1A. O esquemático do circuito digital é apresentado na Figura 41 a seguir. O circuito foi desenvolvido com a ferramenta *Fritzing* (FRITZING, 2020).

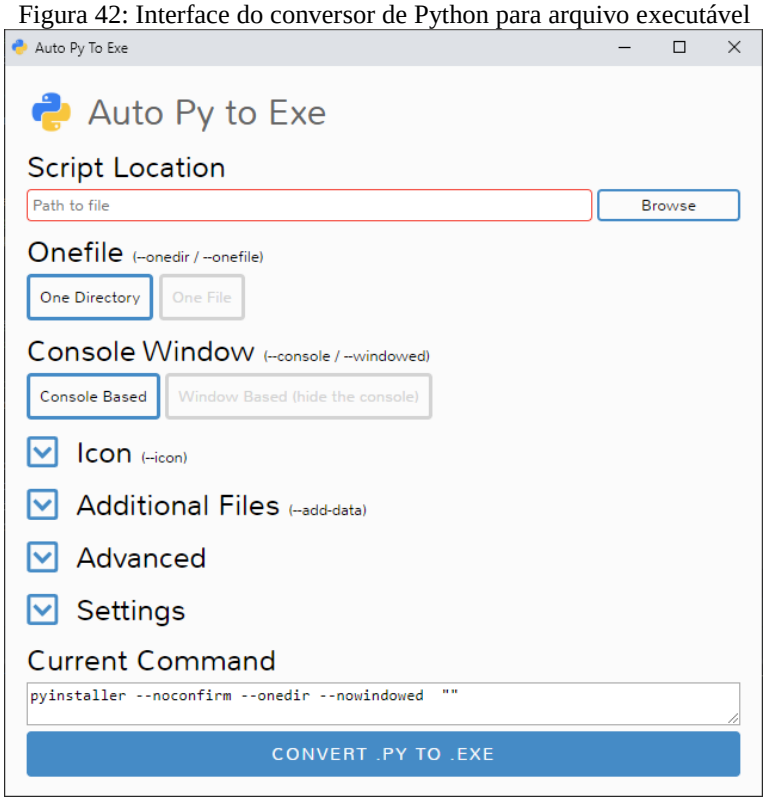
Figura 41: Projeto do circuito digital



Fonte: Próprio autor.

Conversão para arquivo executável: A conversão de um código em *Python* para um arquivo executável é realizada de maneira relativamente simples através do compilador *Pyinstaller*. Uma interface amigável provê todas as funcionalidades do *Pyinstaller* de forma acessível, o *Auto PY to EXE*, utilizado para a compilação do *software* protótipo e possibilitando assim que seja executado em qualquer máquina. Utilizou-se esta interface para a compilação do programa, com a tela principal da mesma sendo apresentada na Figura 42 a seguir.

Figura 42: Interface do conversor de Python para arquivo executável



Fonte: VOLLEBREGT (2020).

6 CONCLUSÕES

O estudo sobre a metodologia aplicada em engenharia de *software*, assim como sua utilização para modelagem dos requisitos e projeto resultou em conhecimentos teóricos e práticos úteis no processo de formação em engenharia como um todo.

Muitos dos preceitos aplicados em engenharia de *software* são derivados e/ou também aplicados em outras áreas da engenharia, como qualidade e processo. O produto final é apresentado como um protótipo, necessitando de montagem do automóvel, assim como do esquema de ligações, embora se apresente também os procedimentos necessários para o funcionamento de todo o projeto.

A utilização da linguagem *Python* na construção do *software* mostrou-se uma solução prática para seu desenvolvimento, embora a biblioteca *Tkinter*, escolhida para a construção da interface não possua a mesma praticidade de outras formas de construção, como APIs próprias para este fim (o *pyGUI* por exemplo).

Entretanto, como um dos principais objetivos estabelecidos nos requisitos era o controle remoto de um dispositivo externo o *Tkinter* tornou-se uma opção das mais interessantes por ser uma biblioteca já pertencente ao *Python* como padrão, com isso possuindo compatibilidade com qualquer hardware que possa utilizar a linguagem. Citando como exemplo o microcomputador *Raspberry Pi* utilizado no projeto, sua integração com *Python* o permite também a portabilidade automática com o *Tkinter*, enquanto que outras bibliotecas e APIs podem não possuir a mesma compatibilidade.

Uma das principais dificuldades deste trabalho foi estabelecimento de uma varredura automática controlável pelo usuário com a impossibilidade do *Tkinter* de alteração nas bordas de *widgets*, trazendo maior complexidade a esta tarefa. A solução de se preencher o plano de fundo dos *frames* pelos quais estavam inseridos os *widgets* foi conseguida após pesquisas em sites de documentação da biblioteca e também fóruns onde outros usuários da biblioteca externaram problemas parecidos. O fato de *Python* ser uma linguagem muito utilizada atualmente com ampla comunidade presente relatando erros e *bugs* também auxiliou na solução de outros problemas que surgiram no decorrer do desenvolvimento do *software*. Finalmente a solução encontrada mostrou-se adequada e compatível com o conceito de Tecnologia Assistiva.

Para o controle do dispositivo automóvel de pequeno porte realizaram-se estudos de eventuais opções e tipos de comunicação, sendo a comunicação serial a encontrada de menor probabilidade de surgimento de erros, porém possuindo o inconveniente de ser necessário o uso presença de cabeamento USB para se estabelecer.

Já para a comunicação *wireless* a utilização de uma rede WLAN mostrou-se de maior valia, com o minicomputador *Raspberry Pi* sendo preferido em comparação à placa *Arduino*, pela possibilidade de conexão WLAN sem necessitar de dispositivos externos e a compatibilidade com a linguagem *Python* e a biblioteca *Tkinter* escolhida. Apesar da falta de testes apresentou-se o projeto para montagem do carro.

Uma das questões a serem levantadas seria o controle do minicomputador à distância e como se daria a execução do *software* tanto no computador onde este efetivamente será executado quanto também no *Raspberry Pi*. Considera-se incialmente que ambos executarão simultaneamente o programa, com as instruções de controle sendo executadas no minicomputador, entretanto são questões a serem verificadas apenas com testes e aplicação efetiva do *software* na prática. Um último ponto a se levar em consideração seria um protocolo de segurança e tratamento de erros para situações não ideais, como a perda de conexão ao se utilizar comunicação *wireless*, retirada ou rompimento do cabo na utilização de comunicação *serial* e como será a reação do *software* a estes eventos, podendo emitir um alerta ou fechamento do menu de controle, por exemplo.

6.1 TRABALHOS FUTUROS

Para a continuidade do trabalho propõe-se:

- Montagem e configuração do kit robótico no LTAD.
- Realização de testes com o público alvo.
- Aplicação do *software* na *web* como um *webapp* se conectando diretamente a uma rede auxiliar com o minicomputador agindo como um dispositivo de *access point*.

7 REFERÊNCIAS

ARANACORP. ***Control a DC Motor with Raspberry Pi***. França, 2018. Disponível em: https://www.aranacorp.com/en/control-a-dc-motor-with-raspberry-pi/. Acesso em: 21 out. 2020.

BERSCH, R., & TONOLLI, J. C. (2006). **Tecnologia Assistiva**. Acesso em: 15 ago. 2020, disponível em < http://www. assistiva.com.br/ >

BERSCH, Rita. **Introdução à Tecnologia Assistiva**. Porto Alegre, 2013. Disponível em: https://ead.ufac.br/ava/pluginfile.php/34072/mod_resource/content/1/Introducao_Tecnologia_Assistiva_Rita%20_1.pdf. Acesso em: 15 ago. 2020.

BERSCH, Rita; SARTORETTO, M. L. **Assistiva: Tecnologia e Educação**. Porto Alegre, 2020. Disponível em: https://www.assistiva.com.br/#STA. Acesso em: 15 ago. 2020.

BOEHM, Barry W. **A spiral model of *software* development and enchancement.** Computer, TRW Defense and Space Systems, EUA, v. 21, ed. 5, p. 61-72, 1988.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do Usuário**, tradução do original em inglês *The Unified Modeling Language User Guide*. 2. ed. Brasil: Elsevier, 2006. 552 p.

BOUNI em COMPONENTS 101. Arduino Uno Pin Diagram. 2018. Disponível em: https://components101.com/microcontrollers/arduino-uno. Acesso em: 19 out. 2020.

BRASIL, SUPREMO TRIBUNAL DE JUSTIÇA, **Inclusão, direito de todos**, disponível em: https://stj.jusbrasil.com.br/noticias/566708254/inclusao-direito-de-todos#:~:text=A%20inclus%C3%A3o%20social%20%C3%A9%20o,Federal%20a%20todos%20os%20cidad%C3%A3os. Acesso em: julho/2020.

BRASIL. (1994). Ministério da Educação. Secretaria de Educação Especial. Política Nacional de Educação Especial. Brasília, Brasil.

BRASIL. **Lei Nº 13.146**. Institui o Estatuto da Pessoa com Deficiência e dá outras providências. , Diário Oficial da República Federativa do Brasil, 6 jul. 2015.

CROSTON, B. ***A Python module to control the GPIO on a Raspberry Pi***. [S. l.], 2016. Disponível em: https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/. Acesso em: 13 out. 2020.

DEVMEDIA. O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML. [S. l.], 2020. Disponível em: https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408. Acesso em: 23 ago. 2020.

EPUSP, USP (São Paulo-SP). **Conceitos de Comunicação Serial Assíncrona**. 2016. Brasil, v. 2016. Disponível em: http://www2.pcs.usp.br/~labdig/pdffiles_2016/conceitos-comunicacao-serial-v2.pdf. Acesso em: 12 set. 2020.

FATIMA, H. The 6 Best Python GUI Frameworks for Developers. [S. l.], 2017. Disponível em: https://blog.resellerclub.com/the-6-best-python-gui-frameworks-for-developers/. Acesso em: 10 nov. 2019.

FRITZING. *Software* Fritzing. [S. l.], 2017. Disponível em: https://fritzing.org/. Acesso em: 13 out. 2020.

GALVÃO FILHO, T**. Deficiência intelectual e tecnologias no contexto da escola inclusiva**. In: GOMES, Cristina (Org.). Discriminação e racismo nas Américas: um problema de justiça, equidade e direitos humanos. Curitiba: CRV, 2016, p. 305-321. ISBN: 978-85-444-1214-5.

GARVIN, David A. **Managing Quality: The Strategic and Competitive Edge**. Estados Unidos: Free Press, 1988. 314 p.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). Censo Brasileiro de 2010. Rio de Janeiro: IBGE, 2012.

LIECHTI, C. ***PySerial's Documentation.*** [S. l.], 2001. Disponível em: https://pyserial.readthedocs.io/en/latest/index.html. Acesso em: 7 dez. 2019.

LINDBLOM, J. ***Serial Communication***. EUA: *Sparkfun*, 2014. Disponível em: https://learn.sparkfun.com/tutorials/serial-communication/all. Acesso em: 14 out. 2020.

LUNDH, F. ***An Introduction to Tkinter***. [S. l.], 2009. Disponível em: https://effbot.org/tkinterbook/. Acesso em: 14 nov. 2019.

MONDAL, S. ***Setting up access point on Raspberry Pi and testing.*** EUA, 2017. Disponível em: https://sriparnaiot.wordpress.com/setting-up-access-point-on-raspberry-pi/. Acesso em: 25 out. 2020.

NOGUEIRA, J., C. JONES AND LUQI. ***Surfing the Edge of Chaos: Applications to Software Engineering***. Command and Control Research and Technology Symposium, Naval Post Graduate School, Monterey, CA, 2000, disponível em: www.dodccrp.org/2000CCRTS/ cd/html/pdf_papers/Track_4/075.pdf. Acesso em: julho/2020.

PRESSMAN, R. **Engenharia de *Software*: Uma abordagem profissional**. 7. ed. São Paulo: AMGH, 2011. 780 p.

PRIKLADNICKI, R., WILLI, R., & MILANI, F. (2014). Métodos Ágeis Para Desenvolvimento de Software. Porto Alegre: Bookman.

RASPBERRY PI FOUNDATION. ***GPIO - Raspberry Pi Documentation***. Reino Unido, 2018. Disponível em: https://www.raspberrypi.org/documentation/usage/gpio/. Acesso em: 20 out. 2020.

RASPBERRY PI FOUNDATION. ***Setting up a Raspberry Pi as a routed wireless access point***. Reino Unido,2018.Disponível em: https://www.raspberrypi.org/documentation/configuration/wireless/access-point-routed.md. Acesso em: 20 out. 2020.

SEDRA, A. Microeletrônica. 5. ed. Brasil: Pearson, 2007. 864 p.

SISSON, Daya. **A educação inclusiva e a Ética da Libertação de Paulo Freire**. Revista Brasileira de Bioética, Brasília, v. 5, 1-4, p. 48-62, 2009.

SOMMERVILLE, I. **Engenharia de *Software***. 9. ed. São Paulo: Pearson, 2011. 529 p.

THE PI***. How to use your Raspberry Pi as a wireless access point***. [S. l.], 2017. Disponível em: https://thepi.io/how-to-use-your-raspberry-pi-as-a-wireless-access-point/. Acesso em: 24 out. 2020.

THOMSEN, A. **Tutorial Módulo Bluetooth com Arduino**. Brasil: Filipeflop, 2015. Disponível em: https://www.filipeflop.com/blog/tutorial-modulo-bluetooth-com-arduino/. Acesso em: 13 out. 2020.

TUTORIALSPOINT. ***Python - GUI Programming (Tkinter).*** [S. l.], 2019. Disponível em: https://www.tutorialspoint.com/python/python_gui_programming.htm. Acesso em: 14 nov. 2019.

VAN ROSSUM, G. Python Geral. [S. l.], 2001. Disponível em: https://docs.python.org/pt-br/3/faq/general.html. Acesso em: 28 ago. 2020.

VIDA, R. **Relacionamento entre classes**. UTFPR, Curitiba. Disponível em: http://www.professorvida.com.br/if62c/material/relacionamentos.pdf. Acesso em: agosto/2020.

VOLLEBREGT, Brent. Auto Py to Exe. [S. l.], 2020. Disponível em: https://pypi.org/project/auto-py-to-exe/. Acesso em: 8 out. 2020.

8 ANEXOS

8.1 ANEXO A: CODIFICAÇÃO DO *SOFTWARE* CONSIDERANDO COMUNICAÇÃO SERIAL

```
import sys,os

from os.path import dirname, realpath, sep, pardir
sys.path.append(dirname(realpath(__file__)) + sep + pardir + sep +
"lib")

from tkinter import *
import time
import serial



global v

v=1000
hbut=420

bgd="#65beff"



root=Tk()
root.title("Carrito")
root.geometry("625x575")
root.resizable(0,0)
root.config(bg=bgd)


class Application:
    def __init__(self, master=None):

        #Definir imagem dos botões
        self.botaojogar = PhotoImage(file="botao_jogar2.gif")
        self.botaosobre = PhotoImage(file="botao_sobre2.gif")
        self.botaovarredura = PhotoImage(file="botao_varredura2.gif")
        self.botaosair = PhotoImage(file="botao_sair2.gif")
        self.menu = PhotoImage(file="menu_principal.gif")


        #IMAGEM

        self.widget0 = Label(master)
        self.widget0["width"]=610
        self.widget0["height"]=407
        self.widget0["image"]=self.menu
        self.widget0["bd"]=0
        self.widget0["bg"]=bgd
        self.widget0.pack()

        # BOTÃO SAIR
```

```
        self.widget1 = Frame(master)
        self.widget1["width"]=125
        self.widget1["height"]=125
        self.widget1["bg"] = "red"

        self.sair = Button(self.widget1)
        self.sair["image"]=self.botaosair
        self.sair["command"] = self.widget1.quit
        self.sair.pack()


        #Botão JOGAR
        self.widget2 = Frame(master)
        self.widget2["width"]=125
        self.widget2["height"] = 125
        self.widget2["bg"] = "red"


        self.jogar = Button(self.widget2)

        self.jogar["image"]=self.botaojogar
        self.jogar["command"]=self.telajog
        self.jogar.pack()

        #Botão SOBRE O JOGO
        self.widget3 = Frame(master)
        self.widget3["width"] = 125
        self.widget3["height"] = 125
        self.widget3["bg"] = "red"


        self.sobre_o = Button(self.widget3)
        self.sobre_o["width"] = 125
        self.sobre_o["height"]=125
        self.sobre_o["command"]=self.telasobre
        self.sobre_o["image"]=self.botaosobre
        self.sobre_o.pack()


        #Botão VARREDURA
        self.widget4 = Frame(master)
        self.widget4["width"] = 125
        self.widget4["height"] = 125
        self.widget4["bg"] = "red"


        self.varredura=Button(self.widget4)
        self.varredura["command"]=self.telavar
        self.varredura["image"]=self.botaovarredura
        self.varredura.pack()



        self.var1()

    def telasobre(self):
```

```
        self.widget1.place(x=475, y=hbut)
        self.widget2.place(x=25, y=hbut)
        self.widget3.place(x=175, y=hbut)
        self.widget4.place(x=325, y=hbut)

        Application4(self)

    def telajog(self):



        Application3(self)

    def telavar(self):


        Application2(self)



    def var1(self):
        self.widget1["bd"] = 4
        self.widget2["bd"] = 0
        self.widget3["bd"] = 0
        self.widget4["bd"] = 0

        #Setando tecla Enter como meio de comando
        self.sair.focus_set()
        root.bind("<Return>", lambda e: quit())

        self.widget1.place(x=475, y=hbut)
        self.widget2.place(x=25,y=hbut)
        self.widget3.place(x=175, y=hbut)
        self.widget4.place(x=325, y=hbut)
        self.widget4.after(v, self.var2)

    def var2(self):
        self.widget1["bd"] = 0
        self.widget2["bd"] = 4
        self.widget3["bd"] = 0
        self.widget4["bd"] = 0

        self.jogar.focus_set()
        root.bind("<Return>", lambda e: Application3())

        self.widget1.place(x=475, y=hbut)
        self.widget2.place(x=25, y=hbut)
        self.widget3.place(x=175, y=hbut)
        self.widget4.place(x=325, y=hbut)

        self.widget4.after(v,self.var3)



    def var3(self):
        self.widget1["bd"] = 0
```

```
        self.widget2["bd"] = 0
        self.widget3["bd"] = 4
        self.widget4["bd"] = 0

        self.sobre_o.focus_set()
        root.bind("<Return>", lambda e: Application4())

        self.widget1.place(x=475, y=hbut)
        self.widget2.place(x=25, y=hbut)
        self.widget3.place(x=175, y=hbut)
        self.widget4.place(x=325, y=hbut)

        self.widget4.after(v,self.var4)

    def var4(self):
        self.widget1["bd"] = 0
        self.widget2["bd"] = 0
        self.widget3["bd"] = 0
        self.widget4["bd"] = 4

        self.varredura.focus_set()

        root.bind("<Return>", lambda e: Application2())
        self.widget1.place(x=475, y=hbut)
        self.widget2.place(x=25, y=hbut)
        self.widget3.place(x=175, y=hbut)
        self.widget4.place(x=325, y=hbut)

        self.widget4.after(v,self.var1)



variable=StringVar()
variable.set("1seg")



class Application2:


    def __init__(self,master2=None):

        new = Toplevel(bg=bgd)
        new.title("Varredura Automática")
        new.geometry("850x400")

        # Impedir o usuário de utilizar a janela principal enquanto
outra janela está aberta
        new.transient(root)
        new.grab_set()
        new.resizable(0,0)
```

```
self.widget7 = Frame(new,bg=bgd)
self.widget7["height"]=30
self.widget7.grid(row=2,column=1,columnspan=1)
self.title = Label(self.widget7,bg=bgd)
self.title["text"] = "MUDE A VELOCIDADE DA VARREDURA"
self.title["font"]=("Comic Sans MS","20")
self.title["pady"]=3
self.title["fg"]="white"
self.title.pack()


self.vazio6 = Frame(new,bg="#65beff")
self.vazio6["height"] = 70
self.vazio6["width"] = 70
self.vazio6.grid(column=2, row=2)
self.vazio7 = Frame(new,bg="#65beff")
self.vazio7["height"] = 70
self.vazio7["width"] = 70
self.vazio7.grid(column=3, row=2)

self.vazio8 = Frame(new,bg="#65beff")
self.vazio8["height"] = 70
self.vazio8["width"] = 70
self.vazio8.grid(row=0, column=9, rowspan=4, columnspan=3)


#Output
self.wtexto = Frame(new)
self.wtexto["height"] = 100
self.wtexto["width"] = 70
self.wtexto["bg"]=bgd
self.wtexto.place(x=590,y=50)
self.texto=Label(self.wtexto)
self.texto["relief"]="solid"
self.texto["font"]=("Comic Sans MS","18")
self.texto["textvariable"]=variable
self.texto.pack()

#Botão +
self.widgetmais=Frame(new)
self.widgetmais["bg"] = "red"


self.plus = PhotoImage(file="70421.gif")
self.mas=Button(self.widgetmais)
self.mas["width"] = 100
self.mas["height"] = 100
self.mas["image"]=self.plus
if v== 1000:
    self.mas["command"] = self.temp1
if v == 1250:
    self.mas["command"] = self.temp2
if v == 1500:
    self.mas["command"] = self.temp3
if v == 1750:
    self.mas["command"] = self.temp4
if v == 2000:
    self.mas["command"] = self.nada
```

```
        self.mas.pack()


        #Botão -
        self.widgetmenos=Frame(new)
        self.widgetmenos["bg"] = "red"

        self.minus=PhotoImage(file="minus.gif")
        self.menos=Button(self.widgetmenos)
        self.menos["width"]=100
        self.menos["height"]=100
        self.menos["image"]=self.minus
        self.menos["relief"]="flat"
        if v==1000:
            self.menos["command"] = self.nada
        if v==1250:
            self.menos["command"]=self.temp0
        if v==1500:
            self.menos["command"]=self.temp1
        if v==1750:
            self.menos["command"]=self.temp2
        if v==2000:
            self.menos["command"]=self.temp3
        self.menos.pack()


        def kitar():
            new.destroy()
        #Botao OK

        self.widgetok=Frame(new)
        self.widgetok["bg"]="red"
        self.widgetok.place(x=210,y=280)
        self.ok=Button(self.widgetok)
        self.ok["width"]=10
        self.ok["height"]=4
        self.ok["text"]="OK"
        self.ok["font"]=("Calibri","14")

        self.ok["command"]=kitar
        self.ok.pack()


        self.can=Canvas(new,height=197,width=197)
        self.can["relief"]="solid"
        self.can["bd"]=2
        self.can["bg"]="white"
        self.can.place(x=520,y=100)
        self.ret=self.can.create_rectangle(0,0,100,100,fill="red")
        self.can.create_line(100,0,100,205,fill="black",width=2)
        self.can.create_line(0,100,205,100,fill="black",width=2)


self.varr()


    # Varredura automática
    def varr(self):
```

```
        self.widgetmais["bd"] = 4
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 0

        self.can.move(self.ret,100,0)

        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)
        self.mas.focus_set()

        self.widgetmenos.after(v, self.varr2)

    def varr2(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 4
        self.widgetok["bd"] = 0
        self.can.move(self.ret,0,100)
        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)

        self.menos.focus_set()


        self.widgetmenos.after(v, self.varr3)

    def varr3(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 4
        self.can.move(self.ret, -100, 0)


        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)

        self.ok.focus_set()

        self.widgetmenos.after(v, self.varr4)

    def varr4(self):
        self.widgetmais["bd"] = 4
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 0

        self.can.move(self.ret, 0, -100)
        print(v)
        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)
        self.mas.focus_set()
        self.widgetmenos.after(v, self.varr5)

    def varr5(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 4
        self.widgetok["bd"] = 0
        self.can.move(self.ret,100,0)
```

```
        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)

        self.menos.focus_set()
        self.widgetmenos.after(v, self.varr6)

    def varr6(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 4
        self.can.move(self.ret,0,100)
        self.ok.focus_set()
        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)

        self.widgetmenos.after(v, self.varr7)

    def varr7(self):
        self.widgetmais["bd"] = 4
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 0

        self.can.move(self.ret, -100, 0)

        print(v)
        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)
        self.mas.focus_set()
        self.widgetmenos.after(v, self.varr8)

    def varr8(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 4
        self.widgetok["bd"] = 0
        self.can.move(self.ret, 0, -100)

        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)
        self.menos.focus_set()
        self.widgetmenos.after(v, self.varr9)

    def varr9(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 4
        self.ok.focus_set()
        self.can.move(self.ret,100,0)

        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)


        self.widgetmenos.after(v, self.varr10)
```

```
def varr10(self):
        self.widgetmais["bd"] = 4
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 0
        self.can.move(self.ret,0,100)

        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)
        print(v)
        self.mas.focus_set()
        self.widgetmenos.after(v, self.varr11)

    def varr11(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 4
        self.widgetok["bd"] = 0
        self.can.move(self.ret, -100, 0)

        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)
        self.menos.focus_set()
        self.widgetmenos.after(v, self.varr12)

    def varr12(self):
        self.widgetmais["bd"] = 0
        self.widgetmenos["bd"] = 0
        self.widgetok["bd"] = 4
        self.ok.focus_set()
        self.can.move(self.ret, 0, -100)

        self.widgetmais.place(x=100,y=120)
        self.widgetmenos.place(x=300,y=120)

        self.widgetmenos.after(v, self.varr)

    def temp0(self):
        global v
        if v == 1250:
            self.temp2()
        if v == 1500:
            self.temp3()
        if v == 1750:
            self.temp4()
        if v == 2000:
            self.nada()
        v=1000
        variable.set("1 seg")
        self.texto.pack()
        self.mas["command"]=self.temp1
        self.menos["command"]=self.nada


    def nada(self):
        pass

    def temp1(self):
        global v
        if v==1250:
```

```
            self.temp2()
        if v==1500:
            pass
        if v==1750:
            self.temp4()
        if v==2000:
            self.nada()

        v = 1250
        variable.set("1,25 seg")

        self.texto.pack()
        self.mas["command"] = self.temp2
        self.menos["command"]=self.temp0


    def temp2(self):
        global v
        if v == 1250:
            pass
        if v == 1500:
            self.temp3()
        if v == 1750:
            self.temp4()
        if v == 2000:
            self.nada()
        v=1500
        variable.set("1,5 seg")
        self.texto.pack()
        self.mas["command"]=self.temp3
        self.menos["command"]=self.temp1


    def temp3(self):
        global v
        if v == 1250:
            self.temp2()
        if v == 1500:
            pass
        if v == 1750:
            self.temp4()
        if v == 2000:
            pass
        v=1750
        variable.set("1,75 seg")
        self.texto.pack()
        self.mas["command"]=self.temp4
        self.menos["command"]=self.temp2


    def temp4(self):
        global v
        if v == 1250:
            self.temp2()
        if v == 1500:
            self.temp3()
        if v == 1750:
            pass
```

```
        if v == 2000:
            self.nada()
        v=2000
        variable.set("2 seg")
        self.texto.pack()
        self.mas["command"]=self.nada
        self.menos["command"]=self.temp3


class Application3:

    def __init__(self, master3=None):

        self.ser = serial.Serial('COM3', baudrate=9600, timeout=1)

        time.sleep(1)
        jogar=Toplevel(bg=bgd)
        jogar.title("Carrito")
        jogar.geometry("400x500")
        jogar.transient(root)
        jogar.grab_set()
        jogar.focus_set()
        jogar.resizable(0,0)

        def xau():
            jogar.destroy()

        self.widget0 = Frame(jogar)
        self.widget0["bg"]="red"
        self.widget0["width"]=100
        self.widget0["height"]=100
        self.widget0.place(x=150, y=325)

        self.ok=Button(self.widget0)
        self.ok["text"]="OK"
        self.ok["font"]="Calibri","14"
        self.ok["width"]=10
        self.ok["height"]=4

        self.ok["command"]=xau
        self.ok.pack()


        self.widget1 = Frame(jogar)
        self.widget1["bg"] = "red"
        self.widget1["height"]=100
        self.widget1["width"]=100


        self.cima=PhotoImage(file="up.gif")
        self.up=Button(self.widget1)
        self.up["width"]=100
        self.up["height"]=100
        self.up["image"]=self.cima
        self.up["command"]=self.motorup
        self.up.pack()
```

```
        self.widget2 = Frame(jogar)
        self.widget2["bg"]="red"

        self.baixo=PhotoImage(file="down.gif")
        self.down=Button(self.widget2)
        self.down["width"]=100
        self.down["height"]=100
        self.down["image"]=self.baixo
        self.down["command"]=self.motordown
        self.down.pack()

        self.widget3 = Frame(jogar)
        self.widget3["bg"]="red"

        self.esq=PhotoImage(file="left.gif")
        self.left=Button(self.widget3)
        self.left["width"]=100
        self.left["height"]=100
        self.left["image"]=self.esq
        self.left["command"]=self.motorleft
        self.left.pack()

        self.widget4=Frame(jogar)
        self.widget4["bg"]="red"

        self.dir=PhotoImage(file="right.gif")
        self.right=Button(self.widget4)
        self.right["width"]=100
        self.right["height"]=100
        self.right["image"]=self.dir
        self.right["command"]=self.motorright
        self.right.pack()

        self.var1()

    def var1(self):

        self.widget1["bd"] = 4
        self.widget2["bd"] = 0
        self.widget3["bd"] = 0
        self.widget4["bd"] = 0
        self.widget0["bd"] = 0

        self.widget1.place(x=150, y=50)
        self.widget2.place(x=150, y=165)
        self.widget3.place(x=35, y=165)
        self.widget4.place(x=265, y=165)
        self.widget0.place(x=150, y=325)

        self.up.focus_set()
        self.up.after(v, self.var2)


    def var2(self):
        self.widget1["bd"] = 0
        self.widget2["bd"] = 0
        self.widget3["bd"] = 4
        self.widget4["bd"] = 0
```

```
        self.widget0["bd"] = 0

        self.widget1.place(x=150, y=50)
        self.widget2.place(x=150, y=165)
        self.widget3.place(x=35, y=165)
        self.widget4.place(x=265, y=165)
        self.widget0.place(x=150, y=325)

        self.left.focus_set()

        self.left.after(v, self.var3)

    def var3(self):
        self.widget1["bd"] = 0
        self.widget2["bd"] = 4
        self.widget3["bd"] = 0
        self.widget4["bd"] = 0
        self.widget0["bd"] = 0

        self.widget1.place(x=150, y=50)
        self.widget2.place(x=150, y=165)
        self.widget3.place(x=35, y=165)
        self.widget4.place(x=265, y=165)
        self.widget0.place(x=150, y=325)

        self.down.focus_set()
        self.down.after(v, self.var4)


    def var4(self):
        self.widget1["bd"] = 0
        self.widget2["bd"] = 0
        self.widget3["bd"] = 0
        self.widget4["bd"] = 4
        self.widget0["bd"] = 0


        self.widget1.place(x=150, y=50)
        self.widget2.place(x=150, y=165)
        self.widget3.place(x=35, y=165)
        self.widget4.place(x=265, y=165)
        self.widget0.place(x=150, y=325)

        self.right.focus_set()
        self.right.after(v, self.var5)


    def var5(self):
        self.widget1["bd"] = 0
        self.widget2["bd"] = 0
        self.widget3["bd"] = 0
        self.widget4["bd"] = 0
        self.widget0["bd"] = 4

        self.widget1.place(x=150, y=50)
        self.widget2.place(x=150, y=165)
        self.widget3.place(x=35, y=165)
        self.widget4.place(x=265, y=165)
```

```
        self.widget0.place(x=150, y=325)

        self.ok.focus_set()
        root.bind("<Return>", lambda e: xau())
        self.ok.after(v,self.var1)


    def motorup(self):
        if self.ser.is_open == False:
                self.ser.open()
        else:
            pass
        self.ser.write(b'0')

        self.ser.close()


    def motordown(self):
        if self.ser.is_open == False:
                self.ser.open()
        else:
            pass
        self.ser.write(b'1')

        self.ser.close()

    def motorleft(self):
        if self.ser.is_open == False:
                self.ser.open()
        else:
            pass
        self.ser.write(b'2')

        self.ser.close()

    def motorright(self):
        if self.ser.is_open == False:
                self.ser.open()
        else:
            pass
        self.ser.write(b'3')

        self.ser.close()


class Application4:
    def __init__(self, master4=None):
        sobre=Toplevel(bg=bgd)
        sobre.title("Sobre o jogo")
        sobre.transient(root)
        sobre.grab_set()
        sobre.focus_set()
        sobre.resizable(0,0)

        self.widget1=Frame(sobre)
        self.widget1.pack()
```

```
        def xau():
            sobre.destroy()


        self.texto=Label(self.widget1)
        self.texto["font"]=("Calibri","14")
        self.texto["fg"]="white"
        self.texto["bg"]=bgd
        self.texto["text"]=("Carrito é um jogo de simulação cujo
objetivo é auxiliar no desenvolvimento\n de crianças portadoras de
deficiências."
                            " Feito por Vanderlei de Oliveira Júnior
como tema\n de seu Trabalho de Graduação com a orientação da professora
Erica Regina Daruichi Ma-\nchado."
                            "\n O objetivo do jogo é controlar um
carrinho com auxílio de recurso de Tecnologia Assis-\ntiva, a varredura
automática, que"
                            " possibilita a substituição do mouse por
dispositivos\n adaptados para mesma função."
                            "\n"
                            "\n"
                            "\n"
                            "     Laboratório de Tecnologia Assistiva
Digital, FEIS-UNESP, 2020")

        self.widget2 = Frame(sobre)
        self.widget2["bg"]="red"
        self.widget2["bd"] = 4
        self.widget2.pack()
        self.botoo=Button(self.widget2)
        self.botoo["width"]=5
        self.botoo["height"]=2
        self.botoo["text"]="OK"
        self.botoo["command"]=xau
        self.botoo.pack()
        self.botoo.focus_set()


        self.texto.pack()



if __name__ == '__main__':
    b = Application(root)
    root.mainloop()
```

```
int E1 = 14;    // velocidade motor 1
int E2 = 15;    // velocidade motor 2
int M1 = 16;    // direção motor 1 - DIREITA
int M2 = 17;    // direção motor 2 - ESQUERDA


char a = 0;

void setup(void) {
  // put your setup code here, to run once:


  int i;
  for (i = 14; i <= 17; i++)
    pinMode(i, OUTPUT);

  Serial.begin(9600);

}

void cima(int a)
{

  analogWrite(E1, a);
  digitalWrite(M1, LOW);
  analogWrite(E2, a);
  digitalWrite(M2, LOW);

  delay (1000);  // Procedimento de parada
  analogWrite(E1, 0);
  analogWrite(E2, 0);


}

void baixo(int a)
{

  analogWrite(E1, a);
  digitalWrite(M1, HIGH);
  analogWrite(E2, a);
  digitalWrite(M2, HIGH);

  delay (1000);
  analogWrite(E1, 0);
  analogWrite(E2, 0);


}

void esq(int a)
{

  analogWrite(E1, a);
  digitalWrite(M1, HIGH);
  analogWrite(E2, a);
  digitalWrite(M2, LOW);
```

```
  delay (1000);
  analogWrite(E1, 0);
  analogWrite(E2, 0);

}

void dir(int a)
{

  analogWrite(E1, a);
  digitalWrite(M1, LOW);
  analogWrite(E2, a);
  digitalWrite(M2, HIGH);

  delay (1000);
  analogWrite(E1, 0);
  analogWrite(E2, 0);

}

void loop(void) {
  // put your main code here, to run repeatedly:

  if (Serial.available()) {

    int val = Serial.read();


    switch (val)

    {
      case '0':
        cima(const int 150);
        break;

      case '1':
        baixo(const int 150);
        break;

      case '2':
        esq(const int 150);
        break;

      case '3':
        dir(const int 150);
        break;

    }
  }
}
```

8.3 ANEXO C: MUDANÇAS NO CÓDIGO EM *PYTHON* PARA COMUNICAÇÃO *WIRELESS*

A codificação do *software* para utilização de comunicação *wireless* via rede WLAN requer uma mudança na classe relacionada ao menu de controle dos motores, removendo-se assim os métodos relacionados ao *pySerial* em detrimento aos métodos contidos na biblioteca *RPi.GPIO*. No início do código tem-se:

```
import sys,os

from os.path import dirname, realpath, sep, pardir
sys.path.append(dirname(realpath(__file__)) + sep + pardir + sep +
"lib")

from tkinter import *
import time
import RPi.GPIO as gpio
.
.
.
```

Na classe relacionada ao menu de controle dos motores:

```
class Application3:



    def __init__(self, master3=None):

        def init():
            gpio.setmode(gpio.BOARD)
            gpio.setup(7, gpio.OUT)
            gpio.setup(11, gpio.OUT)
            gpio.setup(13, gpio.OUT)
            gpio.setup(15, gpio.OUT)
.
.
.
```

Por fim as quatro operações de controle:

```
def motorup(self):
    init()
    gpio.output(7, False)
    gpio.output(11, True)
    gpio.output(13, True)
    gpio.output(15, False)
    time.sleep(1)
    gpio.cleanup()



def motordown(self):
    init()
    gpio.output(7, True)
    gpio.output(11, False)
    gpio.output(13, False)
    gpio.output(15, True)
```

```
    time.sleep(1)
    gpio.cleanup()


def motorleft(self):
    init()
    gpio.output(7, False)
    gpio.output(11, True)
    gpio.output(13, False)
    gpio.output(15, False)
    time.sleep(1)
    gpio.cleanup()

def motorright(self):
    init()
    gpio.output(7, False)
    gpio.output(11, False)
    gpio.output(13, True)
    gpio.output(15, False)
    time.sleep(1)
    gpio.cleanup()
.
.
.
```

O restante da codificação segue idêntico ao anexo A.