Dissertação apresentada à Pró-Reitoria de Pós-Graduação da Universidade Estadual do Maranhão, como parte dos requisitos para obtenção do título de Mestre em Engenharia da Computação do Curso de Mestrado Profissionalizante em Engenharia da Computação e Sistemas no Programa de Pós-Graduação em Engenharia da Computação e Sistemas.

Júlio Cardoso Vidal de Figueiredo

PROJETO DE REESTRUTURAÇÃO DO SOFTWARE GUARÁ

Prof. Dr. Luís Carlos Costa Fonseca

Orientador

Prof. Dr. Henrique Mariano Costa do Amaral

Primeiro Membro da Banca

Prof. Dr. Ivanildo Silva Abreu

Segundo Membro da Banca

Campus Paulo VI

São Luís, MA - Brasil

2019

Figueiredo, Júlio Cardoso Vidal de.

Projeto de reestruturação do software guará / Júlio Cardoso Vidal de Figueiredo.– São Luís, 2019.

85 f

Dissertação (Mestrado) – Curso de Engenharia de Computação, Universidade Estadual do Maranhão, 2019.

Orientador: Prof. Dr. Luís Carlos Costa Fonseca.

1.Software guará. 2.Reengenharia de software. 3.Arquitetura de software. 4.IHC. 5.Sistema de tempo real. I.Título

CDU: 004.416

Elaborado por Giselle Frazão Tavares- CRB 13/665

REFERÊNCIA BIBLIOGRÁFICA

FIGUEIREDO, Júlio Cardoso Vidal de. **Projeto de Reestruturação do Software Guará**. 2019. 78f. Dissertação de mestrado em computação aplicada – Universidade Estadual do Maranhão.

CESSÃO DE DIREITOS

NOME DO AUTOR: Júlio Cardoso Vidal de Figueiredo.

TÍTULO DO TRABALHO: Projeto de Reestruturação do Software Guará.

TIPO DO TRABALHO/ANO: Dissertação / 2019

_____

Júlio Cardoso Vidal de Figueiredo

Johna Baildona 14, Apt 12.

CEP: 40-114, Katowice - Silesia. Polônia

PROJETO DE REESTRUTURAÇÃO DO SOFTWARE GUARÁ

Júlio Cardoso Vidal de Figueiredo

Composição da Banca Examinadora:

Prof. Dr.	Luis Carlos Costa Fonseca	Presidente	UEMA
Prof. Dr.	Henrique Mariano Costa do Amaral	Primeiro Membro	UEMA
Prof. Dr.	Ivanildo Silva Abreu	Segundo Membro	UEMA

UEMA

Dedico este trabalho ao meu filho Murilo e à minha esposa Larissa.

Agradecimentos

Agradeço à minha esposa Larissa, por ter sido sempre minha maior incentivadora em todos meus projetos de vida e especialmente neste momento por ter sido extremamente paciente e compreensiva, permitindo que eu conseguisse trabalhar neste projeto mesmo estando nós dois passando pela incrível, porém cansativa experiência de acompanhar nosso primeiro filho em seus primeiros meses de vida.

A toda minha família por sempre apoiar minha educação e estar presente em todos os momentos importantes da minha vida.

Ao meu orientador Prof. Dr. Luis Carlos Costa, por ter colaborado imensamente guiando os passos, passando tranquilidade e apontando sempre os melhores caminhos e tomadas de decisão para este trabalho.

Ao meu co-orientador Prof. Dr. Alexandre Garcia, por ter dado os direcionamentos iniciais e ter se colocado sempre à disposição para auxiliar e tirar todo tipo de dúvida. Mesmo não tendo sido tão solicitado, foi essencial para o correto entendimento do sistema ao qual este trabalho se refere, tanto por suas explicações em aula, quanto por meio de sua tese de Doutorado.

Aos meus colegas de mestrado que tornaram estes dois anos de muito trabalho, também um período de muita descontração e amizade.

A UEMA por fornecer a estrutura para as aulas e à FAPEMA pelo apoio financeiro.

"And those who were seen dancing were thought to be insane
by those who could not hear the music."
(Friedrich Nietzsche)

Resumo

O Centro de Lançamento de Alcântara (CLA) possui diversos processos que devem ser cumpridos e observados rigorosamente durante uma campanha de lançamento de foguetes. Para tal disponibiliza de diversos sistemas mecânicos, eletrônicos e *softwares* que auxiliam na correta execução destas tarefas. Dentre estes, está o Guará, o *software* responsável por receber os dados de sondagem das torres de vento e calcular os ângulos de ajuste para o lançador de modo a compensar a ação do vento na trajetória do foguete para que esta se desenhe o mais próximo da nominal possível. Este *software* foi desenvolvido há mais de 20 anos e encontra-se defasado em alguns aspectos, por esse motivo, este trabalho propõe uma reestruturação do sistema Guará, com o objetivo de torná-lo um verdadeiro Sistema de Tempo Real, mais robusto, tolerante a falhas, capaz de se adaptar às diferentes situações do CLA e evoluções de equipamentos que aconteceram ao longo do tempo e ainda irão acontecer. Tem por objetivo também tornar sua interface de uso mais amigável para os profissionais da Segurança de Voo e tornar seu código mais fácil de ser mantido e evoluído, através de sua portabilidade para uma linguagem mais moderna e utilizando as melhores práticas do mercado e da Engenharia de Software. Tudo isto sem modificar a sua base de cálculo de ajustes, que deve ser totalmente reaproveitada e apenas portada para o novo código, visto que não foi encontrado nenhum grande problema com esta e ela é confiável.

Palavras-chave: *Software* Guará. Sistemas de Tempo Real. Reengenharia de Software. Reuso de Software. Arquitetura de Software. Interface humano-computador. Engenharia de Usabilidade.

Abstract

Alcantara Rocket Launching Center (CLA) has many processes that must be strictly accomplished during some rocket launching campaign. For this means, they have different mechanical, electronic and software systems that help in correctly executing such tasks. Among those, is Guara Software, which is responsible for receiving data from wind measure towers a calculate adjusting angles for the launcher in such a way that the resulting real trajectory of the rocket can be as close to the nominative trajectory as possible. This software was developed over 20 years ago and now it is outdated in many aspects of its behavior. Because of that this work is intended to propose one full restructuring of the Guara system aiming to make it a proper real time system, more robust and failure tolerant, capable of adapting to different possible situations inside CLA and the evolution of other equipment that have happened along this time and that are still to happen in the future. This work also aims to make the interface more user friendly for professionals who work with flight security and make its code easier to be maintained and evolved, by using a more modern software development language and the best practices in software engineering. This all should be done without changing the calculations inside the system, because these have proven to be completely reliable and no big issue was reported regarding this matter.

Keywords: Software Guará. Real Time Systems. Software Reengineering. Software Reuse. Software Architecture. Human-machine Interface, Usability Engineering.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

NASA	*National Aeronautics and Space Administrations*
EUA	Estados Unidos da América
SVO	Segurança de Voo
RTC	Real Time Computing
UML	*Unified Modelling Language*

Sumário

1 INTRODUÇÃO

O Centro de Lançamento de Alcântara possui diversos sistemas de informação para o auxílio em suas atividades de lançamentos de foguetes. Dentre eles, destaca-se o Guará, um *software* que tem como objetivo realizar análises dos dados de ventos a partir de medições feitas momentos antes de um lançamento e definir se estas condições são propícias para o lançamento ou não. E caso sejam, o Guará apresenta como saída os ângulos que devem ser utilizados para ajuste da Rampa do Lançador Móvel de Foguetes, para que seja compensada a influência dos ventos e o foguete possa seguir sua trajetória o mais próximo possível da trajetória nominal.

Tal software foi desenvolvido em meados da década de 1990 e atualmente encontra-se em necessidade de evoluções e melhorias. Este trabalho tem por objetivo propor uma reestruturação para o citado aplicativo, entregando como resultado final um novo *software*, desenvolvido em uma linguagem mais moderna e com pré-requisitos implementados mais adequados às atividades atuais do CLA. Serão investigados os pontos de falha do sistema atual a partir de entrevistas e lançamentos com os seus usuários (profissionais da Segurança de Voo do CLA) e a

partir destes serão definidas as mudanças e melhorias necessárias a serem implementadas.

1.1 Objetivos

Este trabalho tem como principal objetivo desenvolver uma nova versão do Sistema Guará para que possa funcionar em conformidade com todas as tecnologias disponíveis no Centro de Lançamento de Alcântara, melhorando principalmente suas interfaces com outros sistemas e com o usuário final, porém sem necessariamente alterar as suas bases matemáticas de cálculo do ajuste do Lançador.

1.1.1 Objetivo Geral

Destaca-se como objetivo geral então desenvolver um projeto de Engenharia de Software com o intuito de entregar um novo sistema para cálculo de ajuste de Rampa de Lançador Móvel de foguetes, utilizando como entradas as informações advindas dos diversos dispositivos de medição de ventos, com o intuito de auxiliar o lançamento ou não de foguetes no Centro de Lançamento de Alcântara.

1.1.2 Objetivos Específicos

Para facilitar o desenvolvimento deste trabalho, definiu-se os seguintes objetivos específicos para que se chegue ao resultado final:

- Criar a modelagem técnica em termos de Engenharia de *Software* para o novo sistema a ser desenvolvido.
- Desenvolver um projeto de *software* bem estruturado seguindo as boas práticas mais atuais da Engenharia de *Software*.
- Implementar uma interface mais compatível com as atividades desenvolvidas pelos usuários do sistema e seguindo as melhores práticas de *UX (User Experience)* utilizadas no mercado.
- Implementar um sistema com características de um Sistema de Tempo Real conforme definido pela Engenharia de *Software*.
- Implementar um sistema que possibilite interface com diferentes protocolos de comunicação entre os diferentes *hardwares* existentes no CLA.
- Entregar um sistema que seja possível de ser modificado, atualizado, modularizável e customizável conforme as necessidade do CLA mudem com o passar do tempo.

1.2 Caracterização do Problema

É comum em grandes instituições com mais de duas décadas de existência, que suas atividades utilizem *softwares* que foram desenvolvidos em seus primeiros anos de funcionamento, tornando-se assim dependentes de tais sistemas, que comumente, acabam tornando-se defasado por não sofrer as devidas atualizações e correções necessárias ao longo do tempo. Estes sistemas são então caracterizados

como sistemas legados. A dificuldade em manter e atualizar sistemas legados é grande devido ao fato de normalmente terem sido desenvolvidos em tecnologias antigas e para as quais costuma ser difícil encontrar profissionais capacitados nos dias atuais. Além disso existe um risco alto em refazer algo que já é funcional quando não se tem recursos disponíveis para dedicar exclusivamente a esta tarefa.

O *software* Guará foi desenvolvido em linguagem Visual Basic na década de 90, por pesquisadores brasileiros no IAE (Instituto de Aeronáutica e Espaço), para uso no CLA (GARCIA, 2007) e encontra-se atualmente neste estado de existência: um sistema legado e que não possui pessoal capacitado para que seja evoluído.

A forma como ele é utilizado atualmente é tão incompatível com a necessidade de seus usuários, os profissionais da Segurança de Voo do CLA, que no momento do uso eles precisam utilizar de um software auxiliar para fazer cálculos a partir dos cálculos do Guará e assim conseguir extrair a informação de fato relevante para a autorização ou não do lançamento dos foguetes. A tradução da forma de trabalho descrita acima é que os usuários do Guará precisam utilizar planilhas *Excel* para armazenar dados das medições do Guará e observam esta planilha para fazer a análise final, que poderia ser toda automatizada pelo próprio Guará.

Destacam-se também outros problemas na utilização do Guará atualmente, tais como:

- A incapacidade em receber informações das torres de vento mais novas e modernas do CLA, pois ele foi projetado para trabalhar apenas com as torres existentes na época de seu desenvolvimento e não possui a capacidade de ser expandido para receber novos dados;

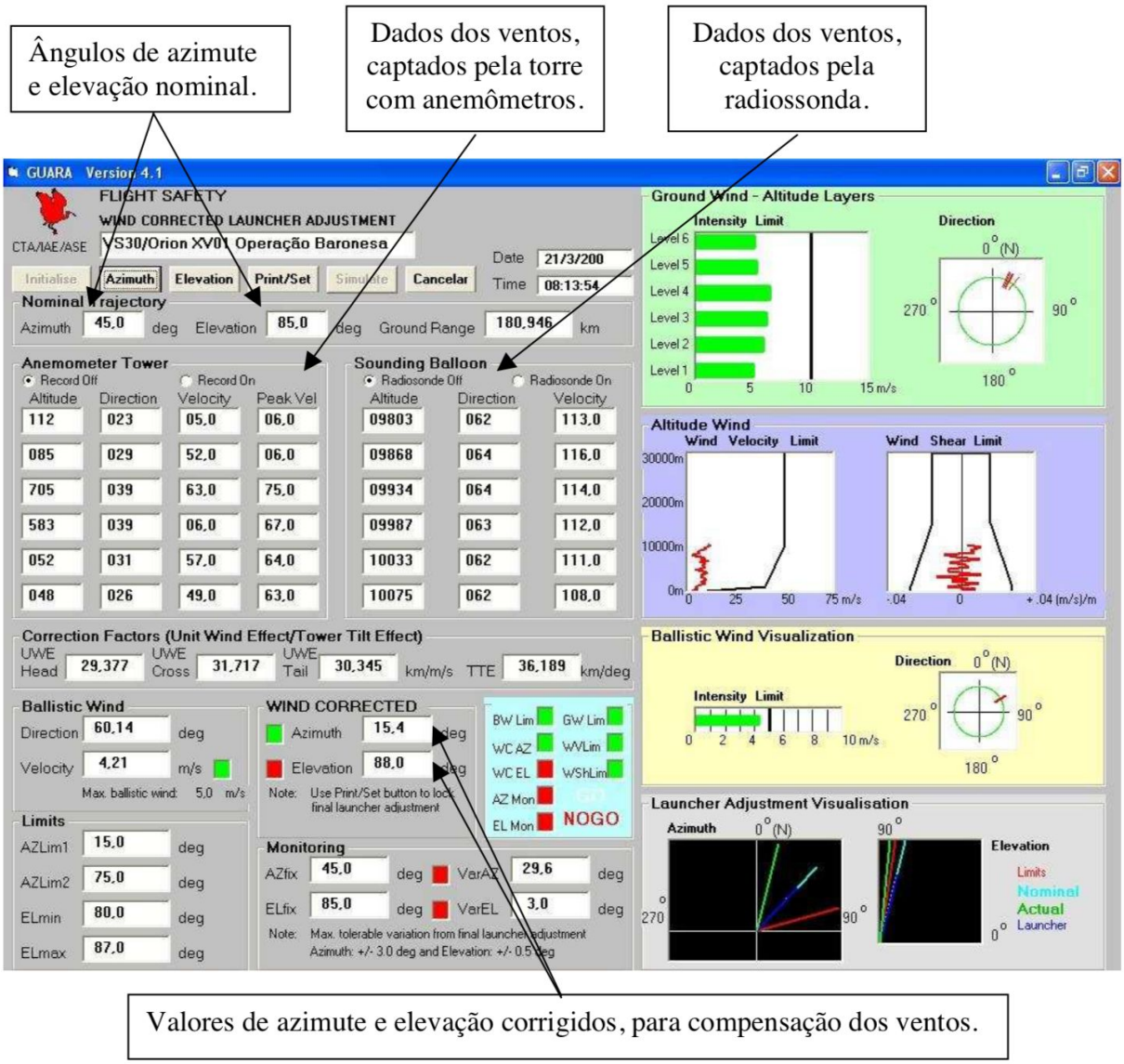
Atualmente o sistema aceita 6 níveis de entrada e a torre nova informa 10 níveis;

- Não é possível definir o valor do ângulo de azimute manualmente, algo que seria relevante para os usuários fazerem testes e verificarem resultados de simulações;
- Para atualizar os dados de azimute fornecidos pelo próprio sistema, é preciso que ele seja reiniciado;
- O sistema trava recorrentemente em momentos críticos de funcionamento.
- Sempre que o sistema precisa ser reiniciado todos os dados de sondagem que haviam sido coletados para o lançamento corrente são perdidos, pois não há uma forma de persistência desses dados;
- Os dados são persistidos somente ao final da sondagem, quando o sistema recebe uma mensagem com um *bit* de término. Sendo assim a cada reinício do sistema é necessário que seja feita uma reconfiguração dos dados da sondagem, o que pode ser demorado;
- Incompatibilidade com os protocolos de comunicação de rede para como os outros sistemas de medição do CLA que o abastecem de informação, sendo assim necessários **emular** *drivers* e dispositivos para que atendam a sua imposição técnica de comunicação exclusivamente por portas seriais;
- Quando ocorre alguma falha, o sistema simplesmente interrompe sua execução, não informando detalhes sobre o erro que causou a falha, o que torna inviável uma investigação para correção de tais falhas;

- *Interface* poluída com informações irrelevantes para o usuário em seu principal momento de uso e sem que seja possível personalizar tal *interface* de maneira a adaptar o sistema para o melhor uso do usuário;
- Dentre outros problemas maiores ou menores a serem investigados mais profundamente no decorrer deste trabalho.

A figura 1 exibe a interface gráfica do Guará como se encontra atualmente com alguns comentários explicativos propostos por Garcia (2007).

Figura 1: Interface do *software* Guará



Fonte: (GARCIA, 2007)

1.3 Justificativa

A importância desta pesquisa é, portanto, desenvolver um novo sistema de cálculo e ajuste da Rampa Móvel do Centro de Lançamento de Alcântara baseado em informações dos dispositivos de medição de ventos, que venha a substituir o

Guará, corrigindo seus erros, processos defasados e adicionando novas funcionalidades, sem introduzir custo algum ao CLA, seja monetário ou de destacamento de seu pessoal interno para tal atividade.

Busca-se assim reduzir custos com equipamentos desnecessários que são utilizados apenas para atender demandas impostas pelas restrições do Guará, como o uso de *software* e *hardware* que simulam as tecnologias por ele exigidas.

Objetiva-se também modernizar as tecnologias utilizadas e principalmente, proporcionar maior agilidade e precisão aos seus usuários no momento de sua atividade sem que precisem fazer uso de outros recursos e *softwares* o que toma tempo e introduz a possibilidade de erro humano no processo que deveria ser totalmente automatizado.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo serão abordados dados de pesquisa bibliográfica com o objetivo de identificar de que maneira o ajuste de trajetória de voo para foguetes não guiados está sendo feito em outros locais do planeta e quais as melhores práticas aplicadas nos sistemas de computador responsáveis por essa tarefa.

Sendo assim foram definidos como principais objetivos dessa consulta identificar 1) alguns dos principais centros de lançamento de foguetes espalhados pelo mundo, 2) como são calculados e informados os ajustes no lançador dos foguetes de sondagem para compensação das perturbações causadas na trajetória a partir do efeito do vento e 3) as responsabilidade do profissional da segurança de voo em lançamentos de foguetes de sondagem.

2.1 Centros de lançamento de foguetes

São oficialmente reconhecidos como centros de lançamento de foguetes no território nacional brasileiro o Centro de Lançamento de Alcântara - CLA, e o Centro de Lançamentos da Barreira do Inferno - CLBI (AEB, 2005).

Os Estados Unidos são reconhecidamente o país mais importante em termos de engenharia aeroespacial e lançamento de foguetes, portanto, faz-se importante tomar conhecimento principalmente da tecnologia desenvolvida por estes. De acordo com a NASA *Sounding Rocket Sciences (2006)* estes são os centros de lançamentos de foguetes de sondagem utilizados pelos EUA desde 1980:

- Wallops Islands, VA (EUA).
- Poker Flat, Alaska (EUA).
- Ft. Yukon, Alaska (EUA).
- Cape Perry (Canadá).
- Andoya (Noruega).
- Ny-Alesund, Spitzbergen (Noruega).
- Estrange (Suécia).
- Ft. Churchill (Canadá).
- Sondre Stromfjord (Greenland).
- Punta Lobos (Peru).
- Alcântara (Brasil).
- Tortugueiro (Porto Rico).

- Kwajalein Atoll (Marshall Islands).
- Kenya (Africa).
- White Sands, NM (EUA).
- Woomera (Australia).

Segundo Garcia (2007) outros centros de lançamento que se encontram em atividade ao redor do mundo são: Anhueng - Coréia do Sul, Balasore - Índia, Thumba - Índia, El Arenossilo - Espanha, Emamshabr - Iran, Haikou - China, Hama-Allepo - Síria, Negev - Israel, Nenoksa - Rússia, Sary Shagan - Cazaquistão, Ryori - Japão e Sonminani - Paquistão.

2.2 Meios de ajuste do lançador

Garcia (2007) explica sobre os lançamentos de foguetes no CLA:

> "Nos lançamentos de foguetes de sondagem não-controlados, realizados no Centro de Lançamento de Alcântara - CLA e no Centro de Lançamento da Barreira do Inferno - CLBI, o cálculo do posicionamento nominal do foguete no lançador é baseado nos estudos da trajetória nominal. De posse de todas as informações disponíveis do vento, em várias camadas de altitude, o operador de segurança de vôo, localizado no centro de controle do centro de lançamento, executa as tarefas necessárias para se obter os valores corrigidos de azimute e elevação, para compensar a influência do vento. Após essa atividade ele informa verbalmente, via telefone, ao operador do lançador, localizado na casamata, os valores finais para o

> posicionamento do foguete. Atualmente, o posicionamento é realizado de forma manual. Contudo, o foguete não é imediatamente lançado, por circunstâncias vinculadas principalmente a forma não automatizada para verificação e correção desse posicionamento. No intervalo de tempo decorrido entre o ajuste final da posição do foguete e o seu lançamento, o vento pode mudar de velocidade e direção, promovendo situação com potencial suficiente para causar desvio significativo na trajetória do foguete. Além da possibilidade de discrepância entre os valores de ajuste do lançador, o operador de segurança de vôo, não recebe a informação do posicionamento atual do lançador, pois essa informação atualmente só é possível de ser obtida pelo operador do lançador localizado na casamata."

Ainda com base na pesquisa de Garcia (2007) que realizou extensivo levantamento sobre como é feito o ajuste no lançador em diferentes centros de lançamento, obteve-se as seguintes informações:

- Segundo Viertotak (2005), *Project Manager of SSC Esrange Space*, em lançamentos de foguetes a partir de Esrange – Suécia, o lançador é ajustado para compensar a influência dos ventos. Essas informações são processadas por programas de computador, mas atualmente em Esrange na Suécia, o ajuste do lançador é feito de forma não automática, ou seja, manualmente.

- Segundo Baxter (2005), membro da *Reaction Research Society* e Vice Presidente da *Pacific Rocket Society*, localizado no Sul da Califórnia dos Estados Unidos da América, o sistema de ajuste de lançadores de foguetes

de sondagem, para compensar a influência do vento, de seu conhecimento, não é feita de forma automática, mas sim mecanizada.

- Segundo Holthaus (2005) *CA Pyrotechic Operator of Friends of Amateur Rocketry* dos Estados Unidos da América, o lançador de foguetes de sondagem utilizado em suas operações de lançamentos, não possui um sistema de correção do ajuste do lançador. Dependendo o tipo de foguete, principalmente quando esse possuir baixa velocidade de saída do lançador, é necessário fazer a correção dos ângulos de elevação e de azimute, pois quanto menor a velocidade de saída do foguete maior será a influência dos ventos na atitude de vôo. Quando necessária tal correção, essa é feita de forma manual.
- Segundo Burrows (2005), *Sr. Scientist & Professor of Department of Astronomy & Astrophysics - Pennsylvania State University* dos Estados Unidos da América, relatou que em todos os lançamentos de foguetes de sondagem que contou com sua participação, a forma de ajuste do lançador foram feitas de forma manual, para compensação da influência dos ventos.

- Segundo Dragoy (2006), *Range Safety Engineer of Andoya Rocket Range* da Noruega, relatou que existem dois lançadores de foguetes na Noruega. O lançador principal fica localizado em Andoya e o outro chamado lançador de "campo" na Ilha de *Spitzbergen* em *Ny-Alesund*. Os dois lançadores não possuem nenhum tipo de sistema automático para o ajuste dos lançadores. Todo o processo de ajuste é feito de forma manual.
- Segundo Jung (2006), *Head of Launch System and Flight Dynamics of Deutsches Zentrum fur Luft und Raumfahrt – DLR (German Aeroespace*

Center), informou que no DLR existem três tipos diferentes de lançadores de foguetes de sondagem não-guiados. Todos esses são comandados de forma manual da casamata ou de *container* instalados nas proximidades do lançador. Todos os lançadores foram fabricados a 30 anos e necessitam de modernização. O último ajuste do lançador é realizado sete minutos antes do lançamento, obedecendo limites pré-determinados de segurança. Jung também relatou que a automatização do lançador pode de fato reduzir o tempo do último ajuste, e conseqüentemente haverá uma redução na área de dispersão do ponto de impacto das partes de um foguete.

A partir dos dados e informações extraídos deste levantamento conclui-se que a melhor opção para o desenvolvimento deste trabalho seria propor um novo sistema que seja completamente alheio ao que se encontra disponível no mercado, especialmente em relação a interface de uso da aplicação. Esta decisão deve-se ao fato de que a principal colaboração que este trabalho pretende oferecer é entregar um sistema que esteja de acordo com as melhores práticas de desenvolvimento e usabilidade utilizadas na atualidade, e para isso não há necessidade de se basear em qualquer destas referências encontradas, mas sim nas boas práticas encontradas na literatura destas áreas específicas.

2.3 Responsabilidades do profissional da Segurança de Voo

Compete ao operador de segurança de vôo, (AEB, 2005, p.12-13):

i) proteger a integridade física das pessoas, as propriedades e o meio ambiente contra danos, que possam ser causados por um veículo lançador, durante a fase de lançamento,

ii) minimizar os riscos de danos decorrentes de um veículo lançador, durante a fase de lançamento,

iii) estabelecer, implantar e manter os planos e regras de segurança de vôo a serem aplicadas na fase de lançamento, compatíveis com a legislação brasileira e com os planos e regras de segurança de superfície aplicáveis,

iv) estabelecer competências, atribuições e requisitos de segurança de vôo para os operadores de veículos espaciais envolvidos,

v) estabelecer e implantar instalações e equipamentos, bem como estabelecer e aplicar procedimentos específicos de segurança de vôo, que possibilitem assegurar que os riscos à segurança de vôo sejam compatíveis com os planos e regras de segurança definido, e que os riscos sejam adequadamente controlados,

vi) identificar os perigos, situações perigosas, avaliar os riscos à segurança de vôo, analisar os riscos dos veículos lançadores, instalações, equipamentos e operações previstas, e eliminar os perigos e situações perigosas ou reduzir os riscos à segurança de vôo a níveis aceitáveis,

vii) administrar os riscos residuais à segurança de vôo inerentes à fase de lançamento,

viii) verificar a aplicação dos planos, regras e procedimentos específicos de segurança de vôo,

ix) aprovar os equipamentos embarcados de segurança de vôo, ou constatar a aprovação realizada por organismo competente,

x) aprovar o plano de vôo de veículo lançador, incluindo trajetória nominal e dispersões.

xi) contribuir para a elaboração e implantação dos planos de segurança quanto às situações de emergência relacionado à fase de lançamento,

xii) informar às autoridades competentes sobre qualquer incidente ou acidente e participar de investigações, documentando as constatações,

xiii) divulgar para outros operadores as lições de segurança de vôo aprendidas;

xiv) estabelecer e operar uma estrutura de segurança de vôo necessária à execução das atividades relativas às suas competências e

xv) elaborar relatórios sobre a aplicação e cumprimento das regras de segurança de vôo, e encaminhá-los ao operador de segurança.

Sendo assim demonstra-se que o papel do SVO é fundamental para o cumprimento com sucesso das missões de lançamento de foguetes de sondagem e o objetivo principal deste trabalho é contribuir com uma ferramenta que otimize e facilite o desempenho das atividades deste profissional em seu momento mais crítico.

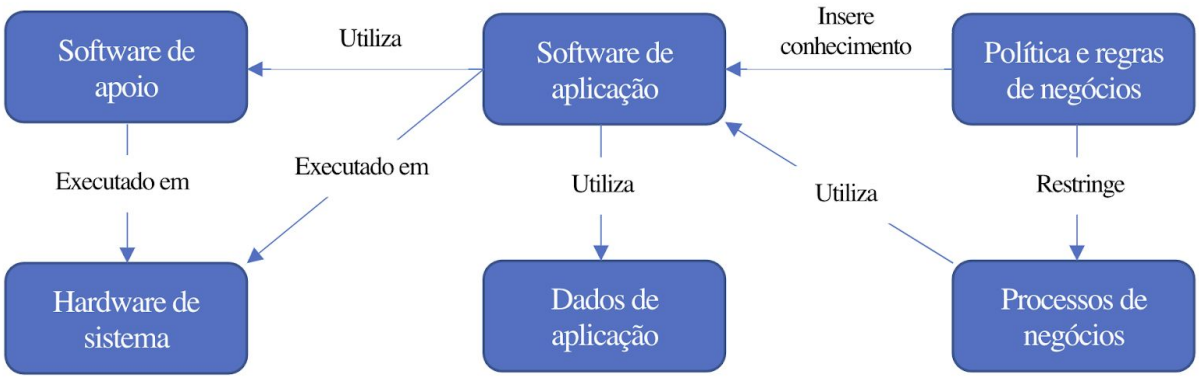
3 FUNDAMENTAÇÃO TEÓRICA

3.1 Sistemas Legados

Em computação, é possível definir um Sistema Legado como uma tecnologia antiga, método, ou aplicação que seja, ou esteja relacionado com uma sistema computacional prévio e desatualizado (Merriam-Webster, 2018). Segundo Bispal et. al (1999) um sistema de informação legado representa um investimento de negócio massivo e de longo prazo. Infelizmente esses sistemas normalmente são frágeis lentos e não extensíveis, ou seja, não propiciam que seja possível realizar alterações de maneira a evoluir suas funcionalidades.

Sommerville (2003) define que sistemas legados são compostos por diferentes componentes que se relacionam entre si, conforme apresentado na figura 2 e descreve estes componentes básicos da seguinte maneira:

Figura 2: Relacionamentos entre componentes de sistemas legados



Fonte: (SOMMERVILLE, 2003)

- **Hardware de sistema:** pode ocorrer que o sistema tenha sido desenvolvido para um hardware específico que já não está mais disponível nos dias atuais.
- **Software de apoio:** muitos sistemas legados podem depender de software específico de um determinado fabricante ou hardware e este software pode também não estar mais disponível no presente momento. Ou o mesmo já não possui o suporte desejado.
- **Software de aplicação:** fornece os serviços de negócio e, geralmente é composto por diferentes sistemas que podem ter sido desenvolvidos em épocas distintas. Comumente o termo "sistema legado" faz referência a este software de aplicação e não ao sistema como um todo.
- **Dados de aplicação:** dados usados especificamente pelos sistemas de aplicação. Como o sistema foi desenvolvido há muito tempo, é comum que estes dados possam estar acumulados em grandes quantidade, além do que ainda é necessário para a execução da

aplicação. Além disso existe boa chance de estarem desatualizados, corrompidos ou duplicados.

- **Processo de negócio:** sistemas legados podem ter processos de negócio associados diretamente a seus componentes, pois o sistema foi desenvolvido para resolver um implementar alguma necessidade da organização. Um exemplo pode ser um sistema que emite a apólice de seguros em uma seguradora. Este sistema está implementando diretamente um processo de negócio.
- **Políticas e regras de negócio:** definições sobre como a empresa deve ser conduzida e possíveis restrições às quais ela deve se submeter. O uso de um sistema legado pode estar acoplado a este conjunto de regras.

Nomear um sistema como legado, em geral significa dizer que ele está em funcionamento há bastante tempo e conseguiu se manter útil por tem uma importância grande dentro da organização, porém já está em necessidade de ser atualizado ou substituído, mas ainda é assim é utilizado. Bennet (1995) lista algumas possíveis razões para isso:

- O sistema funciona de maneira satisfatória e os donos não veem necessidade de trocá-lo;
- O custo para reprojetar o sistema ou trocá-lo por completo pode ser muito alto devido a ele ser grande monolítico e/ou complexo.
- Preparar e configurar um novo sistema pode ser muito custoso em termos de tempo e dinheiro, em comparação com a confortável

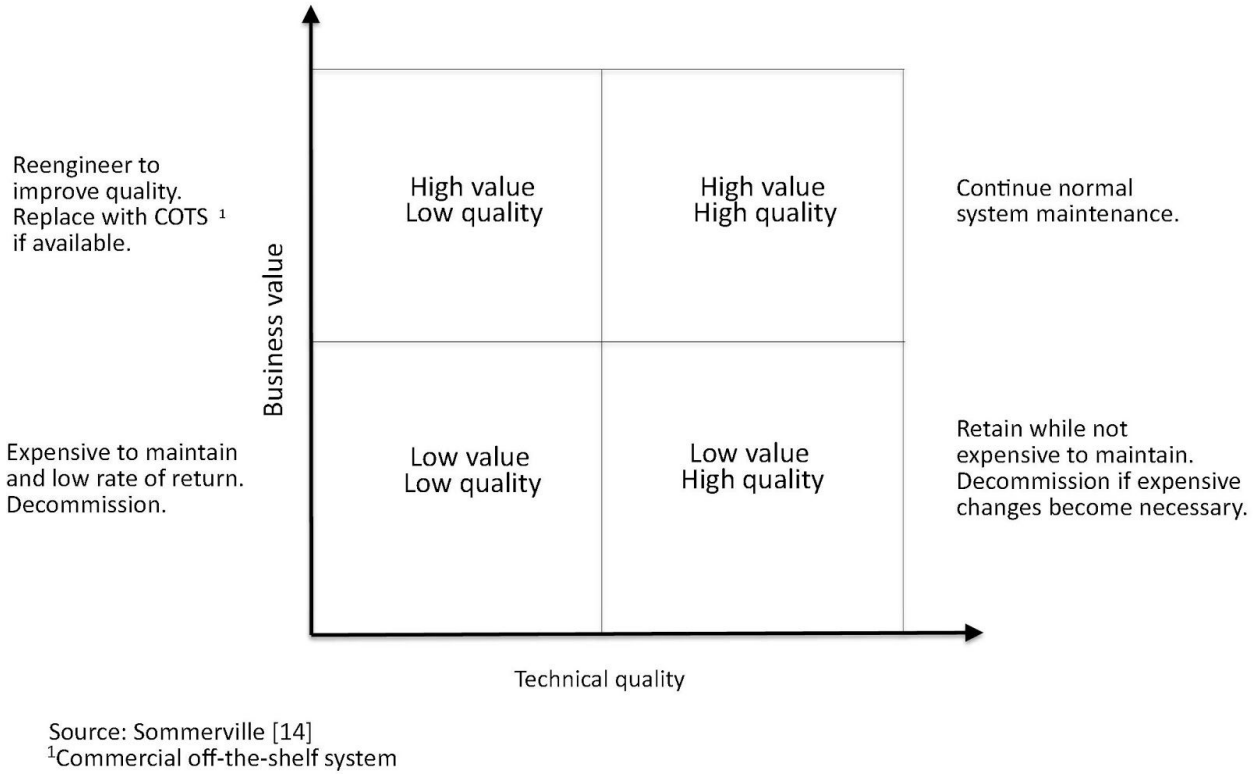
alternativa de manter um sistema que todos conhecem e já funciona de maneira aceitável.

- Possivelmente o sistema é necessário que opere em alta disponibilidade, ou seja, o custo de deixá-lo parado para que seja substituído por um novo ou refeito pode ser muito alto, visto que a organização depende dele funcionando constantemente. Um exemplo claro desta situação poder ser um sistema bancário, que precisa funcionar 24 horas por dia.
- A forma como o sistema funciona não é completamente compreendida pelas pessoas que trabalham com ele, sendo assim fica difícil para que estas pessoas façam modificações no mesmo, pois isto incorreria na possibilidade de adicionar erros às funcionalidades do negócio. Esta situação ocorre geralmente quando as pessoas que desenvolveram o sistema em primeiro lugar já não se encontram mais na organização onde ele opera.
- O usuário final espera que o sistema possa ser facilmente trocado quando estiver desatualizado. O que não é inteiramente uma verdade, visto que toda mudança exige uma fase de adaptação.
- Sistemas novos podem performar de maneira indesejável para os usuários adaptados ao sistema antigo, tanto em termos de regras de negócio, quanto relacionadas a requisitos não-funcionais, como a possibilidade de adicionar brechas de segurança que o sistema legado não possuía, devido ao fato de ter sido projetado para e estar bem maduro com aquele ambiente em que funciona.

A figura 3 demonstra como Bennet interpreta a matriz de decisão para sistemas legados, previamente sugerida por Sommerville, onde cada quadrante representa uma combinação de alto ou baixo de negócio e qualidade técnica. Entregando assim 4 cenários possíveis:

- Baixo valor e baixa qualidade: significa que o sistema é caro para ser mantido e tem baixo retorno. Nesse caso o melhor é descontinuar qualquer esforço que ainda esteja sendo empregado nesse sistema.
- Baixo valor e alta qualidade: nesse caso, o sistema deve ser mantido enquanto sua manutenção estiver barata. Caso necessite de mudanças caras, deve ser descontinuado.
- Alto valor e baixa qualidade: caso mais propício para uma reengenharia do sistema para que seja melhorada a qualidade. Se possível substituir por um sistema “de prateleira”, ou para os padrões atuais, pode-se traduzir como substituir por uma solução facilmente escalável.
- Alto valor e alta qualidade: caso mais desejado. Recomenda-se somente continuar a manutenção do sistema normalmente.

Figura 3: Matriz de decisão de sistemas legados



Fonte: (Bennet, 1995)

3.1.1 Problemas Introduzidos por Sistemas Legados

Bispal et. al (1999) sugere que sistemas legados são considerados por alguns engenheiros de *software* como potencialmente problemáticos por diversas razões. A seguir lista-se algumas dessas razões:

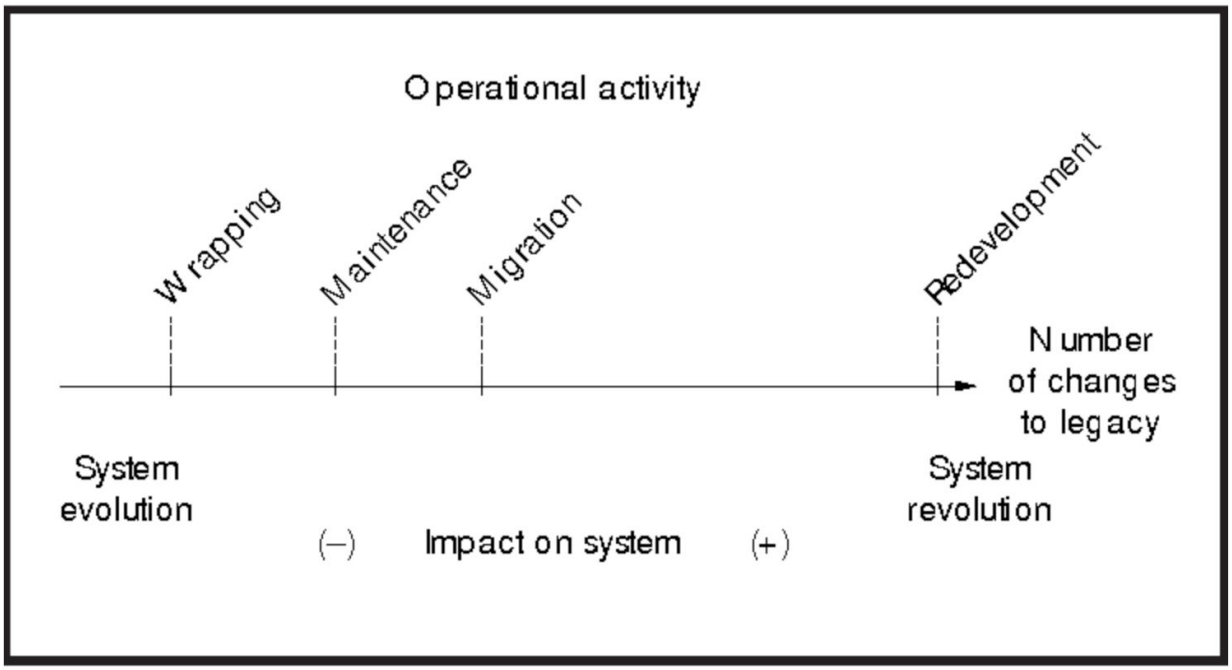
- Se o software só pode ser executado em um *hardware* já antiquado, o custo para manter todo o sistema pode eventualmente superar o custo total para substituição do software e do hardware, a não ser que haja alguma forma de emular o *hardware* antigo ou algum tipo de retrocompatibilidade de *hardware* para que seja possível o uso de um *hardware* novo (LAMB, 2008).
- Esses sistemas podem ser difíceis de manter, melhorar e expandir porque existe uma lacuna de entendimento do sistema generalizada; a equipe que era especialista no sistema envelheceu e provavelmente está aposentada ou não lembra mais precisamente dos seus detalhes e os novos membros que entraram na equipe após ele ter se tornado um "legado" nunca sequer aprendeu sobre ele. Este cenário pode ser ainda piorado pelo cenário comum da perda ou falta de documentação do sistema.
- Sistemas legados podem apresentar vulnerabilidades em sistemas operacionais antigos devido a falta de atualizações de segurança para estes sistemas, que podem nunca ter sido disponibilizadas, ou foram, porém não foram aplicadas. Podem haver também configurações para uso em produção que causem problemas de segurança. Essas questões podem colocar o sistema em risco de ter sua segurança comprometida por ataques externos ou mesmo de internos a partir de pessoas que conheçam estes detalhes

- Integrações com novos softwares também podem ser difíceis porque estes possivelmente usem tecnologias completamente diferentes. Integrações entre diferentes tecnologias é algo bem comum em computação, porém integração entre tecnologias recentes e tecnologias substancialmente antigas não é tão trivial.
- Restrições orçamentárias comumente levam corporações a não atender a necessidade de substituição de um sistema legado. No entanto as companhias em geral desconsideram a crescente economia que pode ser feita com hardware, software e equipe que pode acontecer quando o novo sistema for implantado. Além de subestimar as graves perdas para o negócio que podem ocorrer caso o sistema legado falhe. Uma vez que essas considerações são bem entendidas e o retorno sobre o investimento é comprovado, geralmente o orçamento passa a ser possível para a troca do sistema legado.
- Devido ao fato de que a maioria dos programadores de tecnologias legadas estão se aposentando e os novos profissionais tendem a trabalhar com tecnologias mais recentes, o potencial de força de trabalho para estes sistemas se torna cada vez menor, mais uma dificultando enormemente sua manutenção e evolução.

A figura 4 demonstra o quanto uma mudança em um sistema legado impacta no sistema de acordo com a fase de desenvolvimento em que ela acontece. Caso aconteça nos primeiros estágios, sejam eles *Wrapping* (Entendimento), Manutenção ou Migração, este impacto tende a ser menor e ainda considera-se que esta mudança significa uma Evolução do Sistema. Caso aconteça em qualquer

etapa após estas, está claro que o novo sistema já está em produção e, portanto esta nova mudança significa o "redesenvolvimento" de algo que já havia sido aprovado e entregue, e portanto, considera-se uma Revolução no sistema. Obviamente muito mais impactante em termos de custos para o sistema de uma maneira geral.

Figura 4: Soluções para problemas no desenvolvimento de um sistema e o seu impacto de acordo com a etapa do desenvolvimento.



Fonte: (Bispal et. al, 1999)

3.1.2 Melhorias e Soluções para Sistemas Legados

Existem diferentes abordagens para se tentar resolver os problemas e impactos causados por sistemas legados. Quando é impossível "aposentar"

completamente o sistema devido ao impacto que isto teria no negócio, uma possibilidade é fazer melhorias no próprio sistema para que ele se torne adequado ao uso. Segundo Hein (2006), um sistema é atraente para para reuso se a organização tem capacidade para realizar verificação, validação, testes e histórico operacional do mesmo. Tais capacidade devem ser integradas em várias fases do ciclo de vida do sistema tais como análise, desenvolvimento, manutenção e uso.

Quando não existe total segurança da organização para realizar estas tarefas, possivelmente uma outra solução seja mais interessante. Nesse caso, considera-se também a possibilidade de realizar uma Reengenharia de Software, que segundo Pérez-Castillo (2013) é uma das estratégias mais usadas para evoluir um sistema legado e resulta em produto com a funcionalidade desejada e sustenta a importância do negócio.

Sommerville (2013) apresenta a definição que reengenharia de software se ocupa de reimplementar sistemas legados para que sua manutenção seja mais fácil. Esta reengenharia pode envolver documentar, reestruturar e organizar o sistema, traduzir o sistema para uma linguagem de programação mais moderna e redefinir estruturas e valores de dados do mesmo a fim de adequar-se às novas necessidades.

Para este projeto optou-se por utilizar as técnicas de Reuso e Reengenharia de Software, visto que será necessário entregar um novo software implementado do zero, porém não foi identificada necessidade latente em alterar as principais regras de negócio do Guará, especialmente no que se refere ao cálculo do do ajuste. Este portanto será mantido integralmente como está e apenas portado para uma linguagem de programação diferente.

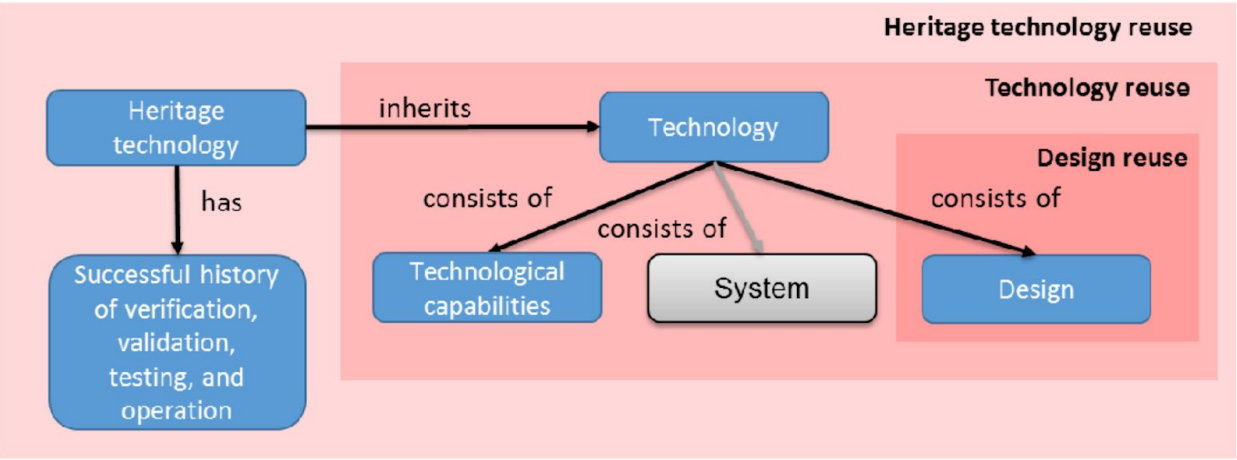
Em conjunto com estas técnicas serão utilizados diagramas UML para definição dos relacionamento entre as classes de um sistema que será desenvolvido

seguindo o padrão da Orientação a Objetos, e também as sequências de atividades do sistema que representam, os fluxos de negócios implementados, com suas possíveis entradas, passos a serem seguidos e esperadas saídas.

O diagrama de reuso de componentes de sistemas definido por Hein (2006) e demonstrado na figura 5 foi usado como base para a decisão de quais partes do sistema deveriam ser reaproveitadas ou não. Segundo este diagrama existem 3 níveis possíveis de reuso de componentes, sejam estes: reuso de tecnologia herdada, reuso de tecnologia e reuso de projeto. O primeiro deve ser escolhido quando se tem um histórico bem sucedido de verificação teste e operações. O segundo reaproveita a tecnologia em si, no caso deste trabalho seria o sistema. E o terceiro reaproveita o projeto.

Para este trabalho foi definido que apenas o projeto será reaproveitado em sua parte de regras de negócio, pois estas, de acordo com levantamento feito, estão muito bem definidas e não há necessidade de serem refeitas. Ademais todas as tecnologias de *Software* envolvidas serão redefinidas para dar lugar a uma infra-estrutura mais moderna, escalável e adaptável.

Figura 5: Várias categorias de reuso.



Fonte: (Hein, 2006)

3.2 Sistemas de Tempo Real

Na Ciência da Computação, o termo **Computação em Tempo Real** (RTC - *Real Time Computing*), ou **computação reativa**, descreve um sistema de *hardware* ou *software* que está sujeito uma "limitação de tempo real, por exemplo a partir de um evento até a esperada resposta (Martin, 1965). Ou seja, diferente do entendimento popular, porém incorreto, de que um Sistema de Tempo Real deve ser um sistema que sempre dá respostas "imediatas" a partir de um dado evento e toma alguma decisão, a verdade é que esse tipo de sistema depende sim de uma limitação temporal, mas não há uma definição de qual seja a duração mínima ou máxima dessa duração definida para que o sistema possa ser classificado ou não como de tempo real.

Programas de tempo real devem garantir uma resposta entregue dentro de um tempo limite, normalmente conhecido como "*deadline*", conforme explica Ben-Ari (1990). Em geral estes limitantes de tempo tendem a ser de fato na ordem dos nanossegundo ou milissegundos, porém está não é uma regra. Por outro lado sistemas que não são necessariamente definidos para se comportar como sistemas de tempo real não possuem esse tipo de restrição, mas comumente tem-se uma expectativa de que respondam em um intervalo semelhante na maioria dos casos.

Martin (1965) também descreveu um Sistema de Tempo Real como “um sistema que controla um ambiente através do recebimento de dados, processamento destes, e retornando o resultado rapidamente o suficiente para afetar o ambiente naquele instante”. Em um aspecto geral pode-se entender portanto um Sistema de Tempo de Real como aquele que utiliza de Computação em Tempo Real para manipular um ambiente, seja ele físico ou virtual, a partir de uma entrada de dados (evento) e em intervalo de tempo limitado e usualmente buscando o imediatismo.

Alguns sistemas utilizados para desempenhar missões em aplicações críticas devem ser implementados como Tempo Real, como sistemas de controle de voo ou sistemas de anti-bloqueio na frenagem de automóveis (Krishna, 2010). Tais sistemas falham em ser de Tempo Real caso não enviem a resposta dentro do intervalo de tempo pré-definido, ou seja, um entendimento muito importante é que estes sistemas devem enviar a resposta dentro do limitante de tempo, independentemente de qualquer sobrecarga que possa estar recebendo.

3.2.1 Classificação de um sistema de tempo real

Um sistema é dito de tempo real se a corretude de sua execução é definida com base não somente na corretude da lógica em si, mas também com base no tempo que a tarefa foi executada. Sendo assim Shin (1994) definiu as classificações possíveis para um sistema de real e sua *deadline* conforme as consequências que significam perder uma *deadline*:

- *Hard* - perder uma deadline significa falha total.

- *Firm* - perdas de deadline com baixa frequência são toleráveis, mas podem degradar a qualidade do serviço do sistema. Além disso um resultado que seja enviado após estourado o prazo da deadline torna-se inútil.
- *Soft* - um resultado enviado após a deadline tem sua usabilidade comprometida, por conseguinte, comprometendo a qualidade do serviço do sistema como um todo. Porém ainda pode ser útil de alguma maneira.

Sistemas de tempo real *Hard* são então desenvolvidos com a obrigatoriedade de que sempre consigam responder a qualquer evento dentro da deadline estabelecida. Tais sistemas são utilizados em situações em que a não resposta a um evento pode causar uma perda grave, seja ela a um estrutura física ou possivelmente por vidas humanas em risco. Alguns possíveis exemplos de sistemas de tempo real *Hard* são sistemas que controlam o motor de um carro ou sistemas de monitoramento cardíaco em hospitais. Muitos destes sistemas são encontrados embutidos em dispositivos e interagindo em baixo nível com o hardware.

Sistemas de tempo real *Firm* tendem a fazer sentido em aplicações que utilizam algum tipo de processamento em paralelo ou concorrente, em que uma tarefa é dividida para que seja executada por diferentes unidades de processamento, otimizando assim o tempo de resposta para o usuário final. Neste tipo de contexto, em geral é aceitável que alguma informação não crítica possa ser perdida, desde que a resposta seja entregue dentro do prazo aceitável e que isto não seja uma via de regra na aplicação. Para este tipo de aplicação costuma-se utilizar o conceito conhecido como *fail fast*, em que é preferível falhar rápido quando uma exceção

ocorre, e entregar a resposta ao usuário do que continuar tentando obter a resposta e manter o usuário esperando por mais tempo do que o desejado (deadline).

Sistemas de tempo *Soft* são comumente utilizados para se resolver o problema do acesso concorrente a recursos e a necessidade de manter atualizados uma séries de sistemas que estejam, de alguma forma, interconectados. Nesse caso em geral é aceitável que algum destes sistemas possa ter uma informação que esteja um pouco defasada em relação a informação verdadeira, que pode já ter sido atualizada em algum outro ponto da rede de sistemas, mas que por algum motivo (ou até mesmo falha) ainda não foi replicada para todos os nós da rede. Os bancos de dado não relacionais, bem como as arquiteturas de sistema modernas baseadas no conceito de microserviços tem forte embasamento nestas premissas. Um outro exemplo bastante claro são sistemas de transmissão ao vivo de vídeo, em que, em geral a perda de alguns pacotes pode ser aceitável pelo fato de que não irá impedir o usuário de assistir o vídeo desejado, apenas resultará em uma degradação na qualidade da imagem (Menychtas 2009).

Para este trabalho foi definido a utilização de uma abordagem *Firm*, visto que a perda de alguma informação ou atraso em entrega de alguma resposta (mesmo que pouco provável, dado o ambiente controlado e seguro em que a aplicacão deve ser executada) não implica necessariamente em um dano crítico que inviabilizaria uma missão de lançamento. Por outro lado a tolerância a falhas, mesmo que existente, deve ser mínima, pois o grau de acerto exigido para o lançamento de foguetes é muito alto e diversas estratégias serão definidas neste trabalho para minimizar quaisquer impactos negativos na operação do CLA que possam ser oriundas de alguma falha do sistema proposto ou de interações com seus perféricos.

3.3 Padrões de Projeto

Na Engenharia de *Software* um Padrão de Projeto, ou *Software Design Pattern*, como são mais conhecidos, significa uma solução genérica e reutilizável para um problema comum em um determinado contexto de modelagem de sistemas computacionais (Gamma 1994). Eles não representam necessariamente uma solução concreta que pode ser traduzida para um determinado *script* ou linguagem de programação, mas sim uma descrição ou arcabouço para como resolver um determinado problema em uma determinada situação. Pode-se dizer que são uma formalização das melhores práticas para resolver determinados problemas.

Padrões de Projeto em Orientação a Objetos geralmente indicam formas de desenvolver soluções no código que resultaram em criação de classes e interação entre objetos através de trocas de mensagens, sem que necessariamente sejam determinados quais serão as classes e objetos finais a serem utilizados (Beck 1987). Este paradigma é provavelmente aquele que possui o maior número de padrões de projetos associados a ele, e alguns deles podem inclusive ser aplicados a outros paradigmas de programação, como a programação funcional. Porém é comum que cada paradigma possua padrões específicos.

Em termos gerais, independente do paradigma de desenvolvimento utilizado, os Padrões de Projeto podem agilizar o processo desenvolvimento do sistema pelo fato de fornecerem uma conjunto de técnicas testadas e comprovadas (Bishop 2012). Desenvolvimento de *Software* efetivo muitas vezes implica em desenvolver um determinado pedaço do sistema de maneira que já esteja preparado para possíveis problemas que possam aparecer no futuro. Quando se escreve código

para sistema, é comum que hajam problemas detectados imediatamente que podem ser logo corrigidos, porém também é muito comum que hajam problemas que só apareçam claramente no futuro e por isso é importante tentar de alguma maneira utilizar soluções já testadas em situações semelhantes. Este é um dos grandes ganhos que se tem utilizando *Design Patterns.*

O reuso dos Padrões de Projeto ajudar a prevenir possíveis problemas conhecidos e também torna a legibilidade do código melhor, pois diferentes programadores ao se depararem com tais estruturas, reconhecem os padrões e com isso já ganha um entendimento sobre qual era a intenção inicial do desenvolvedor que escreveu aquele código em primeiro lugar. Aumento na qualidade de legibilidade do código é muito importante pois torna a manutenção mais fácil e esta é a etapa mais cara do ciclo de vida de desenvolvimento de sistemas computacionais.

Existem também muitos esforços para se definir Padrões de Projetos de acordo com domínios específicos, tornando-os assim mais especialistas e eficazes na solução de problemas. Um exemplo destes são os Padrões de Projetos para interface com usuário (Laakso, 2003). Este é um grupo de padrões que foi definido para uso neste trabalho e será melhor detalhado nas seção seguinte em que será tratado sobre Engenharia de Usabilidade.

Com relação aos padrões de desenvolvimento de código Orientado a Objetos, serão explicados a seguir aqueles de maior relevância para este trabalho:

3.3.1 Padrão *Builder*

O padrão *Builder* tem como objetivo prover uma solução flexível para vários problemas conhecidos na criação de objetos quando se utiliza o paradigma de desenvolvimento Orientado a Objetos. A intenção do *Builder* é separar a criação de um objeto complexo da sua representação (Gamma, 1994).

Os principais problemas resolvidos por este padrão são:

- Como uma classe pode criar, utilizando o mesmo padrão de construção, diferentes representações de um mesmo objeto?
- Como pode uma classe que inclui a criação de um objeto complexo ser simplificada?

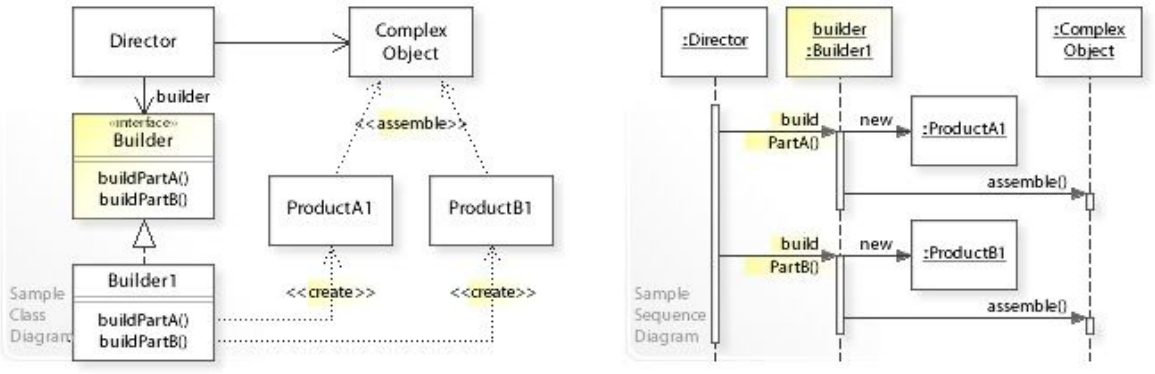
A solução proposta pelo Padrão é baseada na premissa base do desenvolvimento de *Software*: dividir para conquistar. Basicamente o que se faz é impedir que a própria classe seja capaz de construir seus objetos e esta responsabilidade é delegada para uma nova classe, que terá esta única missão de existência e portanto passa a ser mais simples. Costuma-se também utilizar nomes de métodos bem expressivos para facilitar a leitura de quem está utilizando essa nova classe contrutora e assim tornar mais fácil a criação das diferentes representações conforme desejado.

A figura 6 mostra um diagrama de classes e um diagrama de sequência (ambos definidos em *UML*) que ilustram este processo conforme descrito.

No Diagrama de Classes, nota-se que a classe *Director* não é capaz de criar os objetos complexos *ProductA1* e *ProductB1*. Ao invés disso, ela apenas interage com uma interface *Builder*, que é quem conhece os contratos para criação de cada objeto complexo. Porém como é sabido que serão necessárias diferentes representações deste objeto, existe a classe concreta *Builder1* que de fato, implementa os contratos e é capaz de criar cada parte específica dos objetos de maneira isolada.

O Diagrama de Sequência mostra a interação em tempo de execução. Nele podemos ver que o objeto *Director* chama o método *buildPartA* no objeto *Builder1*, este então é capaz de construir e montar o novo objeto *ProductA1*. A seguir o objeto *Director* chamar o método *buildPartB* também no objeto *Builder1*, este então é capaz de construir e montar o objeto *ProductB1*. Finalmente o objeto complexo final é montado com suas partes que foram construidas separadamente.

Figura 6: Diagramas exemplares do Padrão *Builder*.



Fonte: (The Builder Design Pattern)

3.3.2 Padrão *Adapter*

Na Engenharia de Software, o padrão *Adapter* (certas vezes também referenciado como *Wrapper*) é um *Design Pattern* que permite que a interface de uma classe existente possa ser usada como uma outra interface (Freeman, 2004). Usualmente é utilizado para fazer com que classes já existentes possam trabalhar com outras sem que seja necessário modificar o seu código fonte.

A seguir a alguns dos problemas que podem ser resolvidos pelo padrão *Adapter*, de acordo com ws3design (2019):

- Como uma classe que não possui uma interface requerida pelo cliente pode ser reutilizada?
- Como podem classes que possuem interfaces incompatíveis trabalhar juntas?
- Como é possível prover uma interface alternativa para uma classe?

Ou seja, o problema comum a ser resolvido por este padrão é o fato de que comumente uma classe já existente não pode ser reutilizada por não ter uma interface compatível com as necessidade da classe cliente. Para tal problema o padrão *Adapter* descreve como resolver a situação:

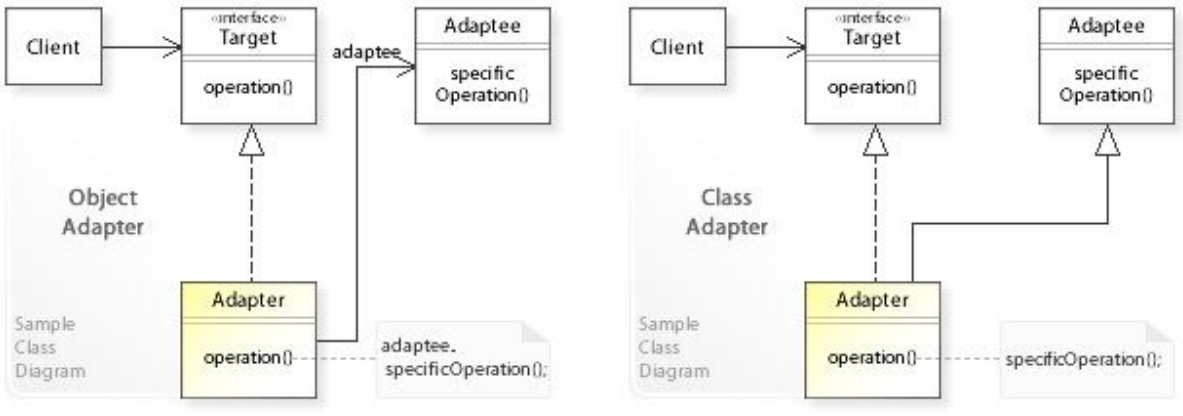
- Define-se uma classe *adapter* separada que converte a interface incompatível da classe existente em uma outra interface que seja em conformidade com as necessidade da classe cliente.

- Cria-se um *adapter* para trabalhar com (reutilizar) classes que não possuem a interface exigida.

O ponto chave deste padrão é que deve-se trabalhar com uma classe *adapter* para que ela interaja com as necessidade da classe cliente, suprindo os pontos de comunicação que não existiam na classe original. Dessa forma a classe cliente acaba por "não saber" se, em dado momento, está interagindo com a classe original ou com a *adapter*.

A classe original que precisa ser adaptada passa então a ser conhecida como "*adaptee"* e a nova interface criada recebe o nome de "*target"*. A figura 6 demonstra essa explicação utilizando um exemplo simples um diagrama de classes UML, onde a classe *Client* que depende da interface *Target* não pode utilizar a classe *Adaptee* diretamente porque esta não possui uma interface compatível com *Target.* Sendo assim, a classe *Client* realiza seu trabalho através de um classe *Adapter* que implementa a interface *Target* e em tempo de execução delega a execução da operação para a classe *Adaptee.*

Figura 6: Diagramas exemplares do Padrão *Builder*.



Fonte: (The Adapter Design Pattern)

3.3.3 Padrão *Strategy*

O padrão *Strategy*, também conhecido como *policy pattern*, é uma estilo de padrão de projeto conhecido como *behavioral software design pattern*, que habilita a seleção de um algoritmo em tempo de execução. Ao invés de implementar um simples algoritmo diretamente, o código recebe instruções em tempo de execução para definir qual algoritmo, dentro de uma família de possibilidades, será utilizado para aquele cenário específico (The Strategy Design Pattern).

A grande vantagem deste padrão é que ele permite que o algoritmo varie de acordo com cliente que está utilizando (Freeman, 2004). Sendo assim torna-se bastante útil em situações em que é preciso definir uma estratégia diferente a ser utilizada para diferentes clientes para um mesmo cenário.

O software Guará, por exemplo, atualmente tem a limitação de não poder utilizar alguns equipamentos novos adquiridos pelo CLA por não ter uma implementação disponível para tais e também não possuir capacidade de adaptação. O padrão *Strategy* consegue resolver este tipo de problema pois ele fornece uma interface única para ser conhecida por seus clientes e as diferentes classes concretas que implementam esta interface ficam a disposição de ser utilizadas conforme necessário. Da mesma forma é possível adicionar novas estratégias sem que seja necessário fazer qualquer mudança na interface ou nos cenários já existentes.

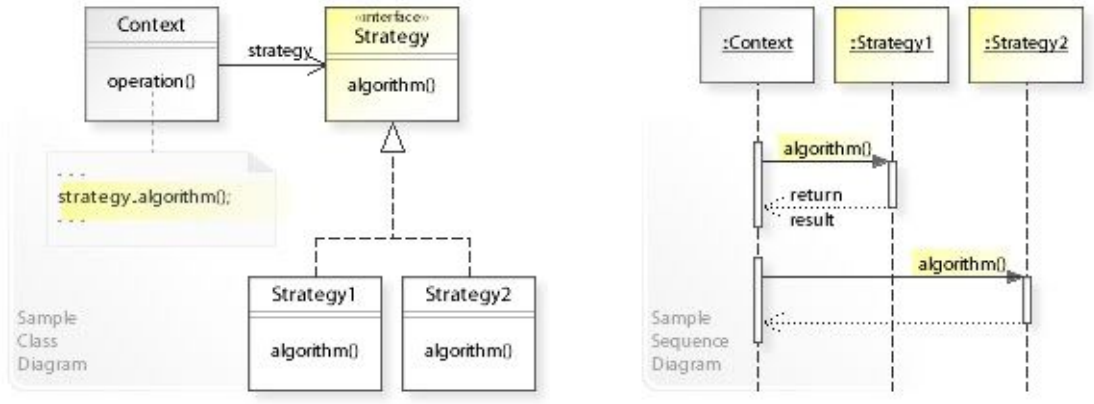
A figura 7 explica essa estrutura e funcionamento através de um diagrama de classes e um de sequência.

No diagrama de classes percebe-se que a classe *Context* não implementa um algoritmo diretamente. Ao invés disso ela se refere, em uma de suas operações, a interface *Strategy*, possui a assinatura do método algoritmo definida. Por fim as classes *Strategy1* e *Strategy2* são as duas classes concretas que assinam a interface e, de fato, implementa o algoritmo conforme sua estratégia específica.

No diagrama de sequência é possível observar o fluxo acontecendo em tempo de execução. Em uma primeira requisição a classe *Context* delega a execução de um algoritmo para a classe *Strategy1*, que faz seu trabalho e devolve um resultado. Em uma segunda requisição, com um contexto diferente, a classe *Context* agora delega a execução do algoritmo para a classe *Strategy2*, que então realizará a tarefa de acordo com as suas regras específicas.

É importante notar que a classe *Context*, em termos de implementação, sequer conhece as classes concretas (como mostra o diagrama de classes). Ao invés diss, ela conhece apenas a interface e passa informações de seu contexto de execução para informar qual das duas classes concretas será de fato utilizada.

Figura 7: Exemplo de diagramas de classe e sequência para o padrão *Strategy*



Fonte: (The Strategy Design Pattern)

3.4 Engenharia de Usabilidade

Engenharia de Usabilidade é um ramo que geralmente preocupa-se com a interação entre humano e computador e, especialmente, em prover interfaces entre humano e computador que tenham alta usabilidade ou que sejam muito amigáveis para o usuário. Ela provê métodos estruturados para que se alcance eficiência e elegância em projetos de interface (Mayhew, 1999).

Diferentes disciplinas de um modo amplo, servem como bases para Engenharia de Usabilidade, como Psicologia, Fatores Humanos e Ciência Cognitiva, mas as fundações teóricas deste campo vem de alguns domínios específicos:

- Percepções humanas e ação;
- Cognição humana;
- Metodologias de pesquisa comportamentais;
- E, em menor relevância, técnicas de análise estatísticas e quantitativas.

Quando se iniciaram as pesquisas e desenvolvimento de trabalhos nesta área, as pessoas responsáveis eram, em geral, profissionais oriundos da Ciência da Computação com alguma ajuda de alguns profissionais da psicologia. Com o tempo, porém, esta tornou-se uma área independente, mais próxima dos departamentos de Ciências Cognitivas e possui suas próprias linhas de pesquisa. Os

profissionais da Ciência da Computação tornaram-se então os maiores clientes desta nova área de estudo (Clark, 2000).

Importante ressaltar que a Engenharia da Usabilidade está mais preocupada em fornecer orientações para melhorias na usabilidade das aplicações e na interação humano-computador, do que em promover mudanças diretamente no *design* dos sistemas. No entanto, os engenheiros de usabilidade acabam se envolvendo nas definições do *design* como produto final do seu trabalho (Nielsen, 2000).

Nielsen (1993) definiu os principais métodos de avaliação de usabilidade:

- Teste de usabilidade;
- Entrevistas;
- Grupos focais;
- Questionários/pesquisas;
- Passo a passo cognitivo;
- Avaliações Heurísitcas;
- Método RITE;
- Análise cognitiva de tarefas;
- Investigação contextual;
- Protocolo de pensamento alto;
- Ordenação de cartas.

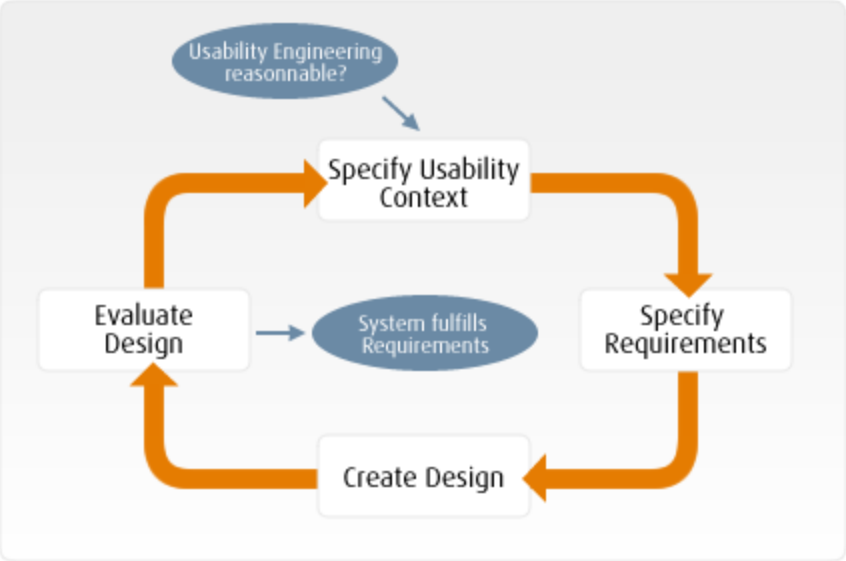
Para este trabalho será utilizado em maior grau de relevância os testes de usabilidade, porém destaca-se também como um possível trabalho futuro aplicação de entrevistas e questionários.

Teste de Usabilidade é definido como a tarefa em que usuários são recrutados e convidados a utilizar o sistema ou protótipo de interface e suas reações, erros, atitudes e auto-relatórios entrevistas são gravados e cuidadosamente observados pelo Engenheiro de Usabilidade. Com base nestes dados, o Engenheiro então recomenda mudanças na interface para melhorar a usabilidade. Esses estudos podem ser analíticos ou empíricos.

A figura 8 demonstra como ocorre o ciclo de vida de um processo de Engenharia de Usabilidade. Nela, é possível observar que a primeira pergunta a ser feita é se "A Engenharia de Usabilidade é razoável?". A partir daí, define-se um contexto de usabilidade a ser especificado. No passo seguinte, definem-se os requisitos. A partir destes, o *design* é criado. Em seguida o *design* é avaliado. E, por fim, verifica-se se o sistema preenche os requisitos.

Interessante notar que mesmo que os requisitos sejam completamente contemplados pelo projeto final, este ciclo não tem fim, pois a Engenharia de usabilidade é definida como um processo constante, que mantem o sistema sempre em evolução.

Figura 8: Definição do ciclo do processo de Engenharia de Usabilidade



Fonte: (Interaction Design Foundation)

Nielsen (2000) definiu ainda 10 heurísticas que tornaram-se a base para a avaliação de qualidade de projetos de software no que se refere ao tema Usabilidade:

- Visibilidade do estado do sistema.
- Reconhecer é melhor que lembrar.
- Compatibilidade entre o sistema e o mundo real.
- Flexibilidade e eficiência para uso.
- Control do usuário e liberdade.
- Design minimalista e estético.
- Consistência e padrões.
- O sistema ajuda o usuário a reconhecer, diagnosticar e se recuperar de um estado de erro.

- Prevenção de erros.
- Ajuda e documentação.

Todos estes critérios foram levados em consideração para o desenvolvimento deste trabalho.

4 METODOLOGIA

Para o desenvolvimento deste projeto será definida a modelagem técnica da Engenharia do Software conforme definido por Pressman (2011) em que serão adotados diagramas UML para definição de classes, objetos e fluxos sequenciais de ações.

A Arquitetura do Software será uma arquitetura de 3 camadas, conforme definido em SILVEIRA (2012), utilizando a linguagem de programação Java, versão 11, o *framework Spring*, banco de dados relacional *MySQL* e o banco de dados *NoSQL Redis* para a camada de *cache*.

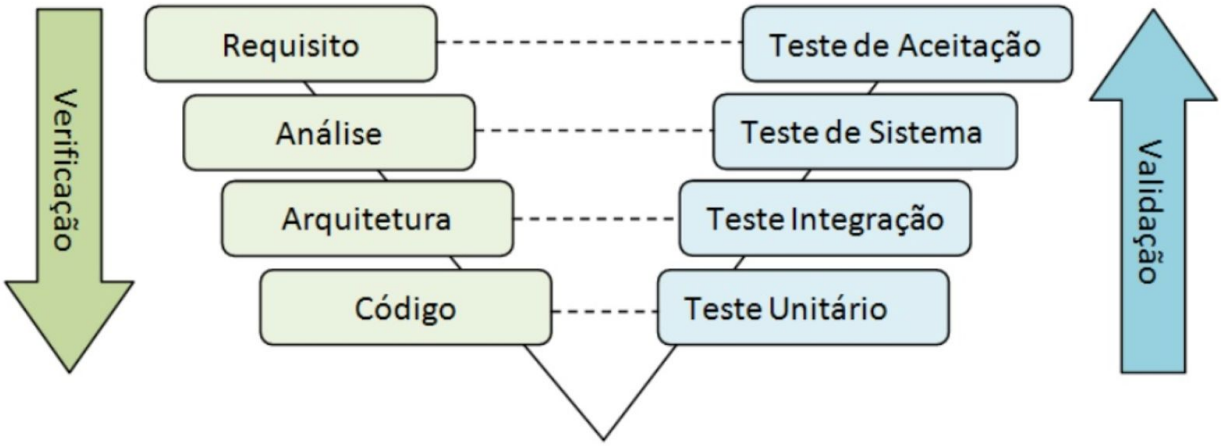
As interações com o usuário e a camada de apresentação do sistema, comumente conhecida como interface será desenvolvida seguindo as heurísticas de Nielsen e o processo de **avaliação heurística** que é um método de avaliação de Interação Humano-Computador criado para encontrar problemas de usabilidade durante um processo de design criativo conforme definido e exemplificado por Barbosa (2010).

A implementação atenderá aos pré-requisitos de um Sistema de Real conforme definido por KOPETZ (1993), como sendo um sistema de controle que deve compartilhar informações críticas entre subsistemas autônomos em um intervalo de tempo definido e de maneira confiável.

Para disponibilizar as interações com os diferentes protocolos e *hardwares* de comunicação em rede do CLA, bem como para tornar possíveis os requisitos de extensibilidade, modularização e customização do sistema o *design* do código seguirá os padrões de projeto *Strategy*, *Adapter* e *Builder*, conforme definidos em GAMMA (1994).

O processo de desenvolvimento do software seguirá o Modelo em V, que pode ser entendido como uma variação do modelo em Cascata, em que são definidos os Requisitos, feita a Análise, definida a Arquitetura e implementado o Código, fechando assim a fase de desenvolvimento ou Verificação do projeto. Para, em seguida iniciar a fase de Validação, em que são então realizados, em sequência, os testes Unitários, de Integração, de Sistemas e de Aceitação, conforme mostra a figura 9:

Figura 9: Modelo em V

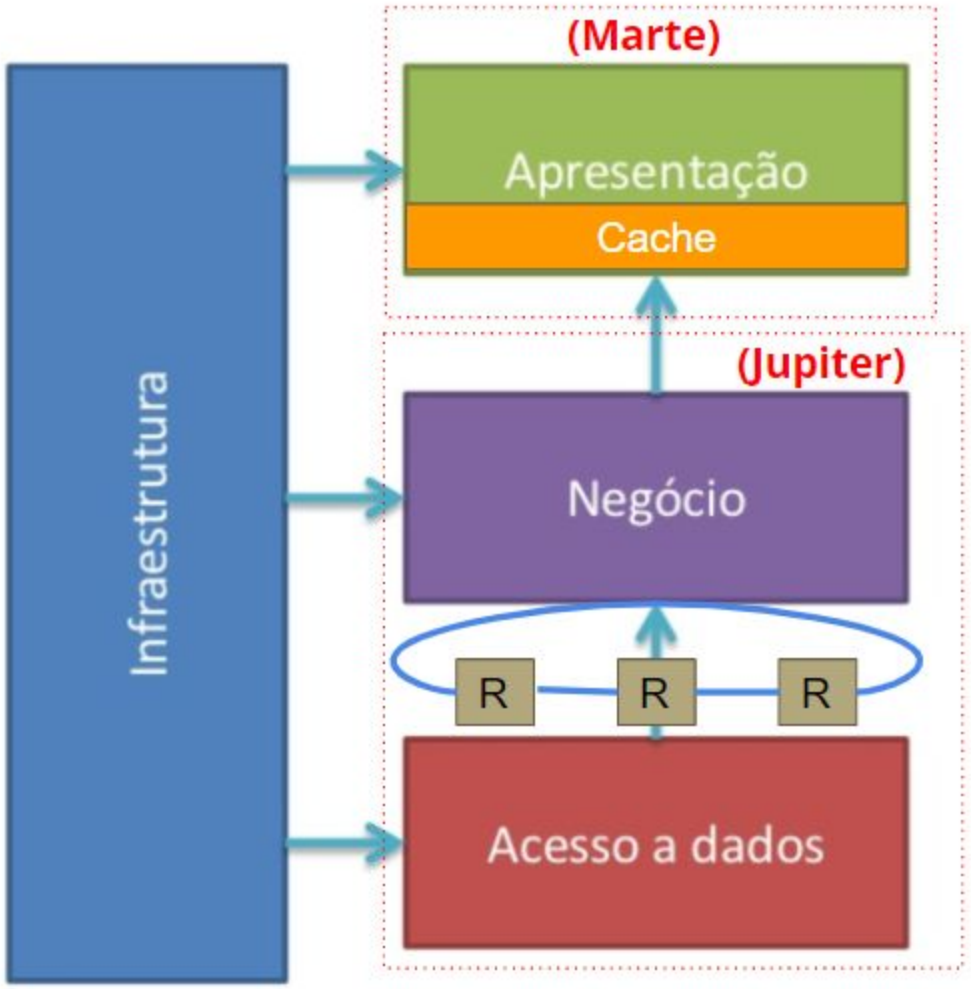


Fonte: (Devmedia)

4.1 Arquitetura do Sistema Proposto

A Figura 10 ilustra como foram divididas as camadas do sistema sistema proposto, formando assim uma solução arquitetada por diferentes componentes trabalhando em conjunto e a seguir estes componentes e suas motivações para que tal formato fosse definido serão explicados em detalhes.

Figura 10: Diagrama Arquitetural da Solução Proposta: *Erucae*



4.1.1 Solução proposta: *Erucae*

A solução como um todo foi nomeada ***Erucae*** e é dividida logicamente em dois módulos com finalidades específicas, sendo estas denominadas **Marte** e **Jupiter.**

4.1.2 Módulo Marte

A aplicação Marte corresponde ao módulo responsável por implementar a camada de apresentação do sistema. Ou seja, ela é a parte que contém todas as interfaces do sistema com o usuario. Sendo assim compreende-se que esta é de vital importância para o sucesso desta proposta, visto que a Engenharia de Usabilidade é um dos pontos chaves para a avaliação de sucesso deste projeto de software.

Conforme apresentado anteriormente, para este módulo, foram escolhidas as tecnologias ***Angular*** e ***Redis***.

Angular é um *framework* de desenvolvimento de aplicações web como foco na parte de interface de usuario, que tem grande poder de facilidade de desenvolvimento de componentes complexos. E este é um dos principais motivos por sua escolha, visto que precisamos de uma camada de interface facilmente adaptável às possíveis mudanças que possam ser solicitadas pelas regras de negócio. Além disso, o Angular é um framework bastante consolidado e com uma forte comunidade de adeptos, tornando assim fácil encontrar informações sobre seu funcionamento, bem como pessoas que possam colaborar em futuras versões.

Redis é um banco de dados *NoSQL (not only SQL)*, o que significa dizer que ele possui características semelhantes a dos bancos de dados tradicionais, porém sem que seja necessariamente um banco de dados relacionais. Este tipo de banco de dados pode ou não possuir características que o divirjam dos bancos de dados tradicionais. No caso do Redis, tem-se um banco de dados com funcionamento baseado em uma lógica chave-valor, que possui alto poder de replicação de dados e

escalabilidade. Esta ferramenta foi selecionada para compor a camada de *cache* de dados do módulo Marte. Sendo assim responsável por guardar as informações em memória sempre que o sistema estiver em uso, para que não haja perda de dados, caso algum problema estrutural ocorra durante uma operação e aplicação Erucae perca contato com qualquer de seus sistemas periféricos. A opção pelo Redis se deu pelo fato de ele ser facilmente configurável para trabalhar desta maneira e também por seu poder de replicação de dados rápida entre os nós, além de fácil integração com as linguagens utilizadas para desenvolvimento: javascript e java.

4.1.3 Módulo Jupiter

A aplicação Jupiter por sua vez implementa a camada que possui as regras de negócio da aplicação, ou seja, este é o local em que todo o processamento do sistema Erucae será de fato processado, sendo o cálculo do ajuste da torre de lançamento, o principal deles. Para esta camada foi definido o uso da linguagem de programação Java, por ser uma plataforma robusta e também fácil de ser adaptada e com bastante adoção pela comunidade de programadores.

Além da linguagem, definiu-se utilizar o framework *Spring* para facilitar o desenvolvimento de toda a lógica de infra-estrutura necessária para que a aplicação funcione. Itens como conexão com bancos de dados e configuração de servidor de aplicação web são então delegados para esta ferramenta, o que facilita tanto o desenvolvimento quanto a manutenção do software, pois torna possível que o desenvolvedor concentre seus esforços apenas nas regras de negócio.

Neste módulo também foi definida camada de cache. Esta implementa com o banco de dados NoSQL *Basho Riak KV*, que também segue uma lógica chave-valor, porém tem maior poder de resistência a particionamento de dados que o Redis. Sendo assim mais adequado para esta camada, que precisa guardar dados mais confiáveis.

Estes dados por sua vez persistidos em eventualmente consultados do banco de dados relacional *MySQL*. Escolhido por ser robusto, simples de usar e amplamente conhecido por profissionais da programação.

Ressalta-se ainda que todas as tecnologias utilizadas sao gratis e de codigo aberto.

5 RESULTADOS

Nesta seção será apresentado um protótipo da solução desenvolvida, especificando a função de cada parte, bem como as soluções apresentadas para os problemas que haviam sido previamente mapeamento no software Guará.

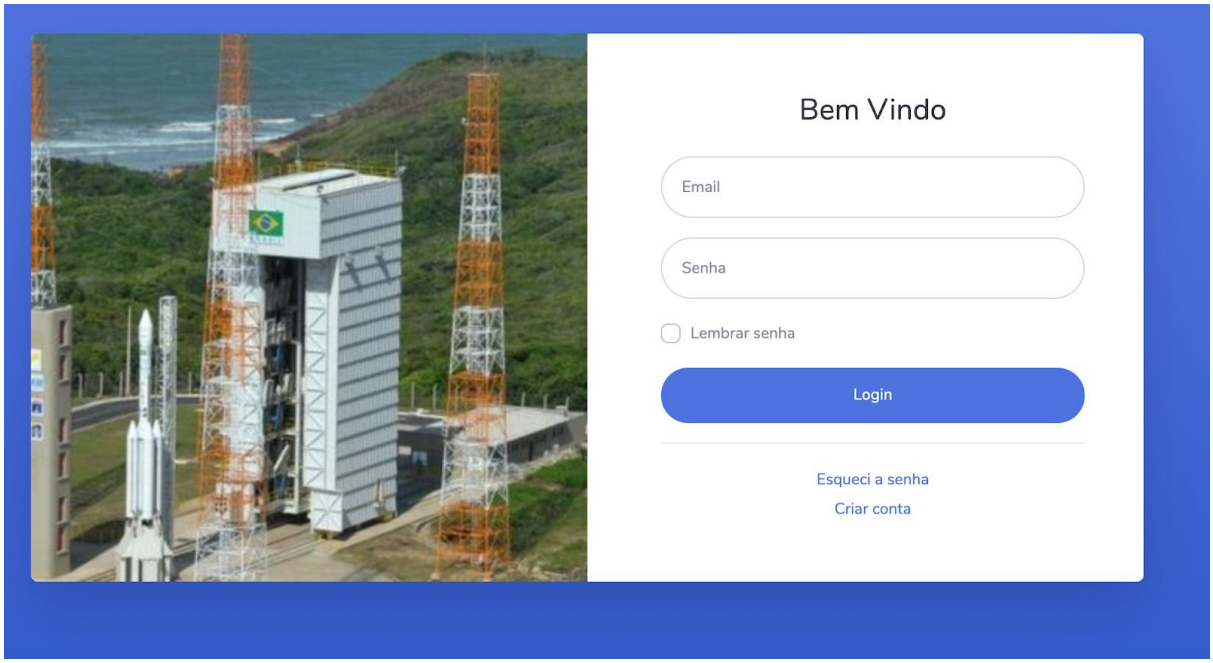
5.1 Protótipo do Sistema Proposto

Nesta seção serão apresentadas telas do protótipo proposto e as motivações por trás das escolhas para cada componente.

A Figura 11 exibe a tela de *Login* da aplicação, que deve ser a primeira tela a ter alguma interação com o usuário do sistema, pois é necessário que qualquer usuário forneça as credenciais adequadas para conseguir utilizar a aplicação.

Destaca-se também nesta tela as opções de Recuperar Senha e Criar Conta. A primeira opção deverá ser utilizada quando o usuário não conseguir lembrar seus dados de autenticação. Neste caso, ele receberá instruções em seu email cadastrado sobre como recuperar sua senha ou criar uma nova. A segunda opção deverá ser utilizada quando o usuário ainda não possuir um login cadastrado na aplicação Erucae.

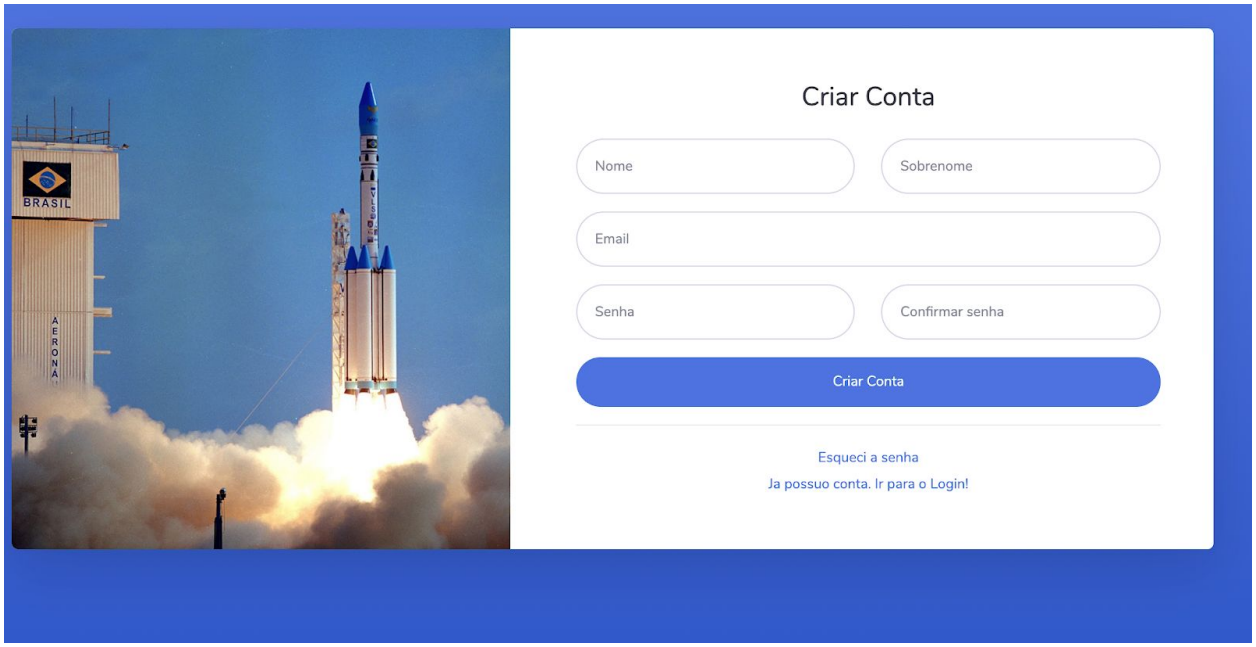
Figura 11: Erucae - Tela de Login



A Figura 12 apresenta uma tela de cadastro de usuarios. Esta deverá ser utilizada quando o SVO ainda não tiver feito o seu primeiro acesso. A princípio definiu-se que para este trabalho haverá apenas um perfil de usuário. Portanto qualquer usuário logado deverá ter acesso a todas as áreas e funcionalidades do sistema. Erucae.

Conforme haja necessidade de adicionar diferentes perfis, esta tela deve ser atualizada para exibir diferentes opções de perfil e o sistema deverá ser atualizado para fazer tais criterios de autorização por recurso. Para tal fim, foram implementados os criterios de autenticação seguindo o padrão de projeto Strategy, que deverá tornar simples a atividade de adicionar um novo perfil e novas estratégias de autorização.

Figura 12: Erucae - Tela de Cadastro



A Figura 13 exibe a Tela de Início, que deve ser a primeira tela a ser visualizada pelo usuário após ter realizado seu login com sucesso no sistema. Esta tela foi desenvolvida objetivando um *design* simples que prioriza o inicio da atividade de medição com recebimento das informações do vento a partir das torres e demais periféricos.

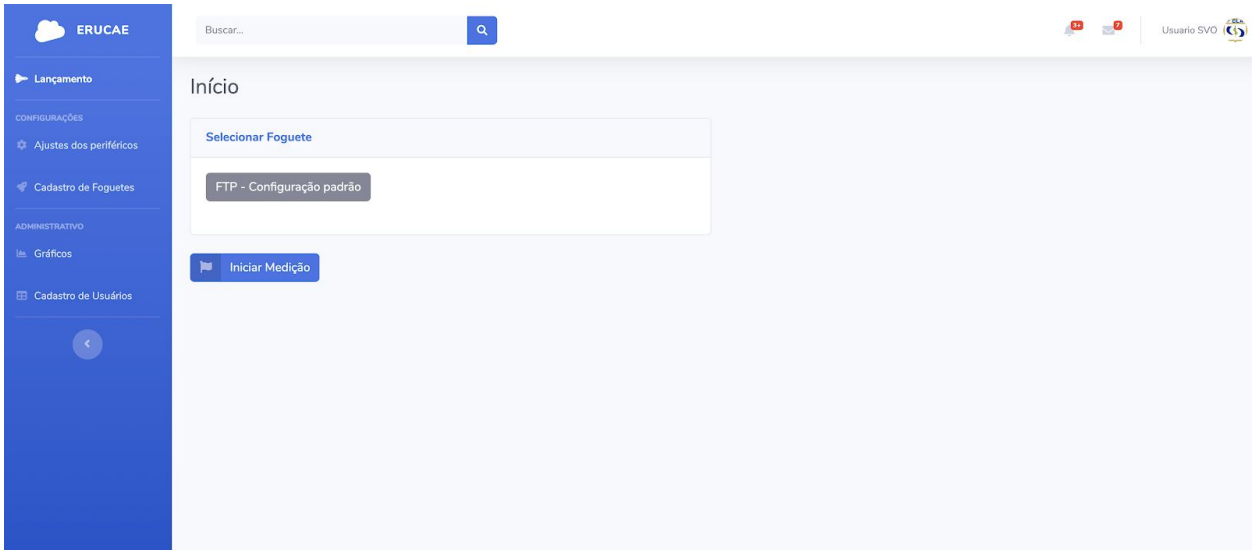
Sendo assim, existem apenas duas ações possíveis a ser tomadas pelo usuário. A primeira é a definição de qual foguete e missão serão utilizados e a segunda é basicamente acionar o botão que inicia a medição.

Para que o foguete e missao estejam disponíveis, eles precisam ter sido cadastrados previamente, utilizando outra tela da aplicação e, tendo assim, suas configurações definidas para uso pelo sistema.

Após clicar no botão de Iniciar Medição, o usuário é direcionado para tela mais importante de sistema. Nesta figura é possível observar também outros

elementos periféricos do *layout* da aplicação, como o menu lateral e a barra de ferramentas no topo. Estes itens serão detalhados mais adiante.

Figura 13: Erucae - Tela de Início



A Figura 14 exibe a tela de Lançamento, onde o profissional da SVO provavelmente passará a maior parte do seu tempo enquanto estiver utilizando a aplicação, em momentos de campanha.

O principal objetivo a ser alcançado nesta tela foi de entregar uma imagem limpa e focada apenas nos dados mais relevantes para o momento do lançamento, evitando que o SVO corra risco de se distrair com informação irrelevante para sua atividade principal. Esta decisão foi tomada baseada no problema levantado pelos usuários que identificaram o Guara como tendo uma interface de uso muito poluida.

Para tal, definiu-se então que no topo da tela fossem exibidos os dados da missão, como o nome do foguete, missão, data e hora. Estas informações são

consideradas o “cabeçalho” da medição para o sistema Erucae, pois juntas compõem um identificador único para cada medição a ser realizada.

Ainda no topo foi posicionado um botão de Gerar Relatório. Esta é uma *feature* nova, proposta pelo sistema Erucae, que tem como função tornar possível que o SVO gere relatórios da missão a qualquer momento. Porém preferencialmente deverá ser mais utilizada ao final de cada missão, com objetivo de criar um recorte histórico para análises futuras. Vale ressaltar que todos os dados coletados e gerados durante uma medição serão automaticamente salvos em banco de dados persistente pelo sistema Erucae. Portanto estes relatórios podem de fato ser gerados em qualquer momento e, também, podem ser definidos novos relatórios de acordo com as necessidades do CLA.

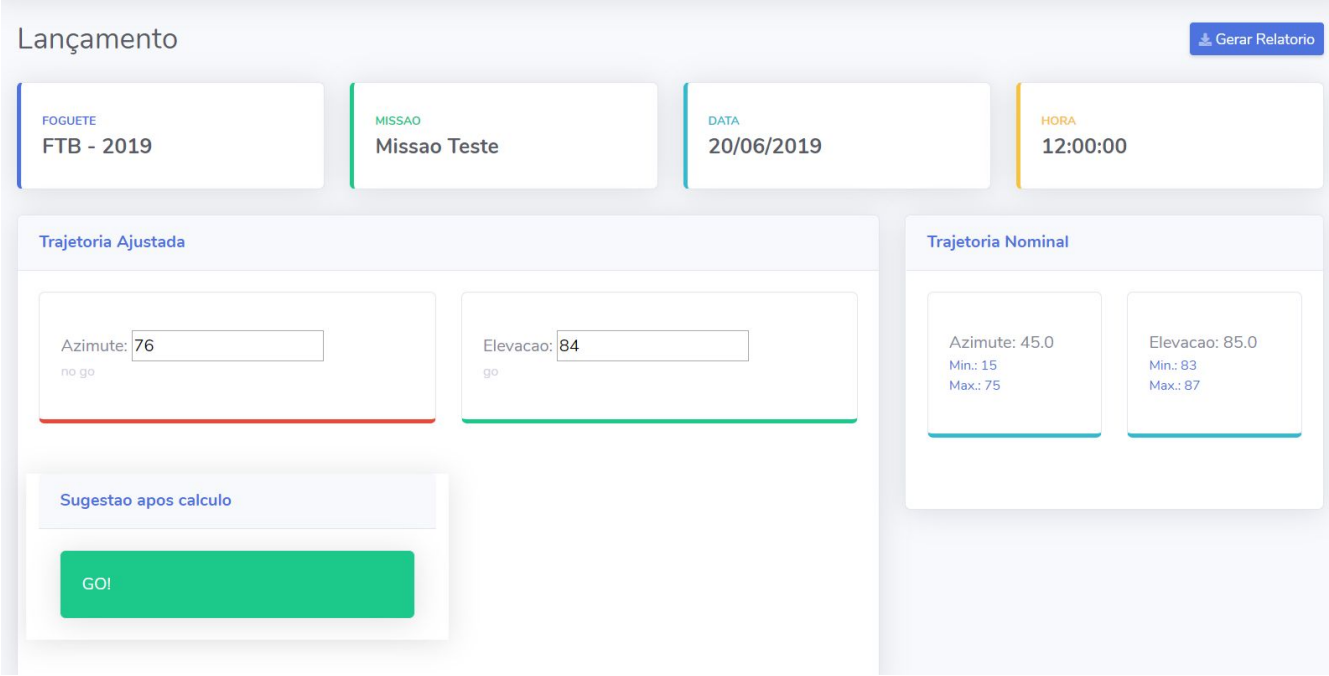
A seguir foram posicionados os dados referentes a trajetoria. Em um primeiro painel, tem-se os dados da Trajetória Ajustada, onde ve-se os ângulos de azimute e elevação propostos pelo sistema após o cálculo e um botão que ficara verde em caso de “GO” e vermelho em caso de “NO GO”. Muitas informações que poderiam ser consideradas relevantes, como os valores individuais de cada torre de vento por camada foram excluídos desta visualização porque considerou-se que seria mais efetivo exibir apenas os dados práticos para tomada de decisão final. Estes dados porém, podem ser consultados a qualquer momento na tela de relatórios.

A direita do painel de Trajetória Ajustada, foi posicionado o painel de Trajetória Nominal. Em uma posição de menor destaque, porém ainda facilmente visível. Este painel exibe as informações padronizadas do foguete: os valores padrão dos ângulos de azimute e elevação; e seus valores máximos e mínimos. Estas informações são importantes ainda porque o SVO pode tomar uma decisão

diferente da proposta pelo Erucae (GO ou NO GO), caso considere que mesmo estando fora dos limitantes, o ajuste ainda está em um raio seguro.

Destaca-se ainda que os valores propostos de ângulos ajustados sao campos editáveis dentro desta tela. Sendo assim, é possível que o segurança de voo, faça o ajuste manualmente, conforme levantando como um desejo, durante a fase de levantamentos de requisitos.

Figura 14: Erucae - Tela de Lançamento

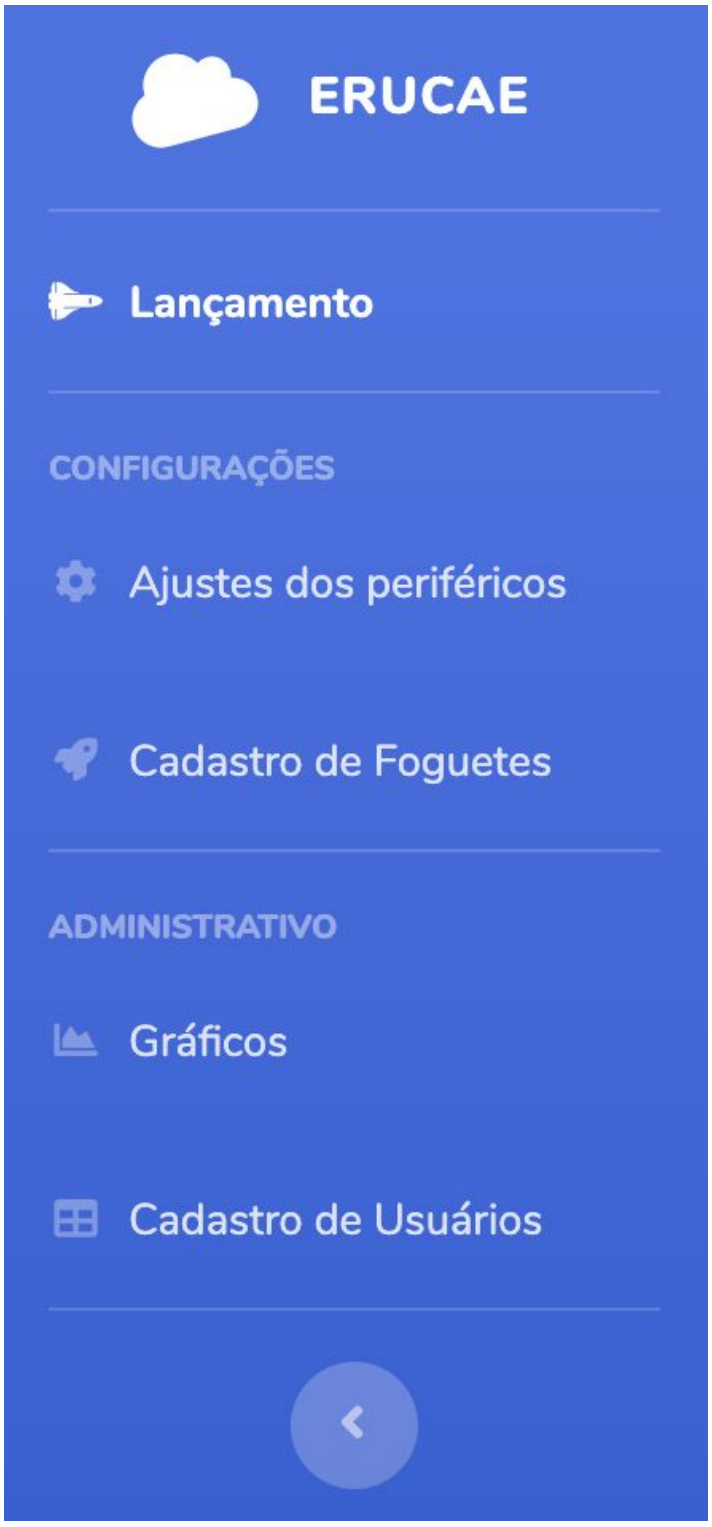


A figura 15 mostra o menu lateral em destaque para que sejam apresentados todos os itens possíveis de ser selecionados para uso no Erucae. Sendo eles:

- **Lançamento**: tela para acompanhamento da medição e ajuste dos dados durante uma campanha de lançamento de foguete.

- **Ajustes dos periféricos**: tela para cadastro e edições de todos os equipamentos que precisam interagir com o Erucae, como torres de ventos e radares.
- **Cadastro de Foguetes:** Tela para cadastro de foguetes e missões com suas devidas configurações.
- **Graficos:** Tela para acompanhamento de gráficos e informações omitidas da tela de lançamento, como dados individualizados dos periféricos.
- **Cadastro de Usuários:** Tela básica para cadastro e edição de usuários.

Figura 15: Erucae - Menu Lateral



A Figura 16 apresenta a possibilidade de comprimir o menu. Função pensada para que este item ocupe menor espaço quando o SVO estiver em missão, para que assim possa ter foco total na tela de lançamento.

Figura 16: Erucae - Menu lateral comprimido

A figura 17 exibe a Barra de Utilidades, que possui um campo de busca para que o usuário acesse qualquer página do sistema através de busca textual; um ícone para email que podera integrado com o email pessoal do SVO; um ícone de mensagens para notificações da aplicações; e as informações do usuário logado.

Figura 17: Erucae - Barra de utilidades

5.2 Análise do Sistema Proposto

Nesta seção serão analisados os resultados encontrados com o sistema proposto, com o principal objetivo de identificar o quão próximo este resultado está do que foi levantado como desejo dos usuários da aplicação.

A principal vertente de análise definida foi a usabilidade da aplicação para com o usuário final, seguindo as práticas recomendadas de Engenharia de Usabilidade e adequação das soluções propostas aos problemas levantados.

Apresenta-se a seguir novamente a lista de problemas encontrados durante a fase de levantamento de requisitos, porém, agora acompanhados pelos soluções implementadas ou propostas. A tabela 1 exibe estas informações agrupadas para fácil visualização:

Tabela 1: Comparação entre problemas levantados e soluções propostas.

Problema	**Solução**
Desenvolvido em Visual Basic na década de 90. Linguagem antiga e defasada.	Utilizada a linguagem Java versão 1.8. Linguagem moderna e robusta em constante evolução. Além de auxílio de *frameworks* modernos, como *angular* e *spring*.
Falta de pessoal capacitado para evoluir o sistema.	Tecnologias modernas propostas tornam mais fácil encontrar pessoal capacitado.
Profissionais da SVO precisam utilizar planilhas *excel* em conjunto com o Guará para armazenar dados das medições durante a campanha.	Todas as medições são automatizadas e apenas as informações relevantes são exibidas para o usuário na tela principal de uso durante a campanha.
Incapacidade em receber informações das torres de vento mais novas.	Sistema adaptável para receber novas integrações com novos periféricos e também passível de edições nós periféricos já existentes.
Não é possível definir o ângulo de azimute manualmente.	Campos de informações relevantes editáveis manualmente de acordo com o perfil do usuário.
Para atualizar os dados de azimute	Sistema implementado seguindo os

fornecidos pelo próprio sistema é preciso que ele seja reiniciado.	padrões de alta disponibilidade, de modo que não precise de nenhum tempo desligado para recarregar informações importantes.
Sistema trava recorrentemente em momentos críticos do funcionamento.	Implementadas melhores práticas de tolerância a falhas e estratégia *fail-fast*, em que, ao identificar uma falha grave impossível de ser contornada, o sistema rapidamente se coloca em um estado básico de funcionamento, sem perda de informações relevantes e evitando riscos de travamento.
Sempre que o sistema precisa ser reiniciado, todos os dados de sondagem que haviam sido capturados são perdidos.	Implementadas duas estratégias de cache em memória e também persistência em disco, para evitar perda de dados, mesmo em situações de falhas gravíssimas.
Dados são persistidos somente ao final da sondagem.	Dados mantido em memória conforme são recebidos e persistidos para o disco simultaneamente, porém em estratégia de processamento concorrentemente para evitar sobrecarga e lentidão na interface para o usuários

Incompatibilidade com os protocolos de comunicação de rede para com os outros sistemas de medição do CLA.	Interfaces de comunicação desenvolvidas são agnósticas ao protocolo de comunicação utilizado por outros sistemas. Resultado alcançado devido ao uso de interfaces de programação que isolam esta camada da aplicação (infra-estrutura), das camadas relacionadas a regras de negócios.
Quando ocorre alguma falha, o sistema interrompe sua execução, sem dar detalhes do erro, tornando impossível uma investigação das causas do problema.	Implementadas melhores práticas de rastreio de erros através de logs ricos em informações relevantes para tais investigações.
Interface poluída e com informações irrelevantes para o usuário em seu momento principal de operação.	Interface proposta é limpa e segue as boas práticas de Engenharia de Usabilidade para melhor interação humano-computador.

Um próximo passo, posterior a este trabalho, deverá ser a implantação do sistema no CLA, e validação destas soluções propostas com o usuário final.

6 CONCLUSÃO E TRABALHOS FUTUROS

Sistemas legados invariavelmente possuem limitações que prejudicam o desempenho da atividade para a qual eles são importantes, e muitas vezes estas atividades cruciais para a organização da qual o sistema faz parte. Justamente por esse motivo não é possível simplesmente desfazer-se do sistema, visto que ele tem grande importância para o negócio.

Em muitos casos as organizações acabam ficando na difícil situação de não poder se desfazer do sistema e também não poder aplicar as melhorias desejadas por conta de uma série de limitações conforme já explicado neste trabalho.

Para a execução deste projeto foram utilizadas técnicas da Engenharia de Software como Reegenharia e Reuso de Software, Modelagem Técnica de Software, Projeto de Arquitetura de Software e uso de Padrões de Projeto para código orientado a objetos. Além do uso de técnicas de avaliação da interação humano-computador para definir a interface gráfica com foco em proporcionar uma boa experiência para o usuário.

Foi definido que seria desenvolvida uma nova proposta para esta solução, sem necessariamente tomar como base outras já existentes, com a intenção de propor algo realmente novo neste mercado. Este objetivo foi considerado como alcançado com sucesso pois o protótipo segue somente as melhores práticas utilizadas na indústria da Engenharia de Software e na disciplina de Engenharia de Usabilidade. Sendo assim, ele é agnóstico, em termos de usabilidade e performance, aos sistemas legados do domínio em que está envolvido. Apresentado

soluções focadas somente no que foi levantado como requisitos desejados pelo usuários final.

6.1 Trabalhos Futuros

Destaca-se que no referente a parte técnica da Engenharia de Software, ainda há mais espaço para aprofundamento no referencial teórico especialmente nos temas sobre Sistemas de Tempo Real, Modelagem Técnica de Sistemas Orientados a Objeto, Diagrams UML, Arquitetura de Software em 3 camadas e interação humano-computador utilizando heurísticas de Nielsen e técnicas de UX, para que estes sejam aplicados de uma maneira ainda melhor nas próximas fases do ciclo de vida desta solução, visto que é entendido que a vida de um Software é um ciclo contínuo e infindável de adaptações e melhorias.

Propõe-se ainda que seja entregue uma documentação técnica mais completa. Incluindo documentos de modelagem do sistema, com diagramas UML e de arquitetura, bem como manuais de uso e instalação. Com o objetivo de facilitar a manutenção futura do sistema Erucae.

Com relação a análise da solução, propõe-se que sejam realizadas mais estratégias de validação, utilizando as heurísticas de Nielsen e os fundamentos da Engenharia de Usabilidade, especialmente que seja feito uso das ferramentas de entrevista com o especialista e questionário. Ou seja, após implantação do protótipo no CLA, devem ser executadas simulações guiadas com o profissional da Segurança de Voo para que este possa ter suas impressões sobre o sistema e

responder a entrevista e o questionário. A partir destes dado e, de acordo com a necessidade, novas adaptações e correções devem ser aplicadas ao sistema Erucae.

Propõe-se ainda que o sistema passe a ser utilizado em suas primeiras campanhas reais, em paralelo ao Software Guará atual, para que sejam tomadas as impressões comparativas entre os dois, coletados mais dados para melhorias e também para que o usuário torne-se confortável com o uso do Erucae em uma situação real, para que torne-se possível e suave a transição para o uso somente desta solução e o eventual desligamento do Software Guará, em detrimento do Erucae.

REFERÊNCIAS

BARBOSA, Simone Diniz Junqueira & SILVA, Bruno Santana. **Interação Humano-Computador**. São Paulo: Elsevier, 2010.

BAXTER, Kevin. **Launch Rail**. [mensagem pessoal] Mensagem recebida por: <alexandregarcia@iae.cta.br>. em: 18 maio 2005.

Beck, Kent; Cunningham, Ward (September 1987). ***Using Pattern Languages for Object-Oriented Program***. OOPSLA '87 workshop on *Specification and Design for Object-Oriented Programming*. Retrieved 2006-05-26.

Ben-Ari, M., "**Principles of Concurrent and Distributed Programming**", Prentice Hall, 1990. ISBN 0-13-711821-X. Ch16, Page 164

BENNET, K. **Legacy Systems**. *IEEE Software*, pp. 19-73, Jan. 1995.

BISBAL, J.; LAWLESS, D.; WU, B.; GRIMSON, J. **Legacy Information Systems: Issues and Directions**. *IEEE Software*. **16**: 103–111. doi:10.1109/52.795108. 1999.

Bishop, Judith. "C# 3.0 **Design Patterns: Use the Power of C# 3.0 to Solve Real-World Problems**". C# Books from O'Reilly Media. Retrieved 2012-05-15. If you want to speed up the development of your .NET applications, you're ready for C# design patterns -- elegant, accepted and proven ways to tackle common programming problems.

BURROWS, David. **Information About Launch Rail for Sounding Rocket.**

[mensagem pessoal] Mensagem recebida por: <alexandregarcia@iae.cta.br>. em: 31

março 2005.

CLARK, D. (2000). **Instructional System Design**.
http://www.nwlink.com/~donclark/hrd/sat.html

DRAGOY, Petter. **Information about launch rail adjustment**. [mensagem pessoal]

Mensagem recebida por: <alexandregarcia@iae.cta.br>. em: 12 set 2006.

Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert (2004). "**Head First Design Patterns**" (paperback). O'Reilly Media: 244. ISBN 978-0-596-00712-6. OCLC 809772256. Retrieved 2013-04-30.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES John Vlissides. **Design Patterns: Elements of Reusable Object-Oriented Software**. EUA: Addison-Wesley, 1994.

GARCIA, Alexandre. **AUTOMATIZAÇÃO APLICADA A LANÇADORES DE FOGUETES DE SONDAGEM PARA COMPENSAÇÃO DA INFLUÊNCIA DOS VENTOS**. Tese apresentada à Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, para a obtenção do título de Doutor em Engenharia Mecânica na área de Projetos. Guaratinguetá, 2007.

Hein, A. M. **Heritage Technologies in Space Programs - Assessment Methodology and Statistical Analysis**, PhD thesis Faculty of Mechanical Engineering, Technical University of Munich. 2006

HOLTHAUS, Mark. **Launch Rail**. [mensagem pessoal] Mensagem recebida por: <alexandregarcia@iae.cta.br >. em: 23 maio 2005.

KOPETZ, Hermann & GRUNSTEIDL, Gunter. TTP - A Protocol for Fault Tolerant Real-Time Systems. **FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing**. IEEE, 1993.

JUNG, Wolfgang. **Automatic Launcher Adjustment for Unguided Sounding Rocket**. Carta enviada por: Wolfgang Jung (German Aerospace Center – DLR), para:
Alexandre Garcia (CTA/IAE) em 11 de Janeiro de 2006.

Krishna Kant (May 2010). *Computer-Based Industrial Control*. PHI Learning. p. 356. ISBN 9788120339880. Retrieved 2015-01-17.

Laakso, Sari A. (2003-09-16). **"Collection of User Interface Design Patterns"**. University of Helsinki, Dept. of Computer Science. Retrieved 2008-01-31.

LAMB, John. **Legacy systems continue to have a place in the enterprise**. *Computer Weekly*. 2008. <http://www.computerweekly.com/feature/Legacy-systems-continue-to-have-a-place-in-the-enterprise>. Acessado em 27 de julho 2018.

MARTIN, James (1965). ***Programming Real-time Computer Systems***. Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4. ISBN 978-0-13-730507-0.

Mayhew, Deborah J. (1999). ***The usability engineering lifecycle: a practitioner's handbook for user interface design***. San Francisco, Calif.: Morgan Kaufmann Publishers. ISBN 978-1558605619.

Menychtas, Andreas; Kyriazis, Dimosthenis; Tserpes, Konstantinos (July 2009). "**Real-time reconfiguration for guaranteeing QoS provisioning levels in Grid environments**". *Future Generation Computer Systems*. **25** (7): 779–784. doi:10.1016/j.future.2008.11.001.

MERRIAM-WEBSTER <http://www.merriam-webster.com/dictionary/legacy>. Acessado em 22 de junho de, 2018.

NASA SOUNDING ROCKET SCIENCE (Estados Unidos da América). Operations/Launch/Ranges. Disponível em: <http://rscience.gsfc.nasa.gov/opsrang.html>. Acesso em: 18 ago. 2019.

NIELSEN, J. (1993). **Usability Engineering**. New Jersey: Academic Press.

NIELSEN, J. (2000). Designing Web Usability: **The Practice of Simplicity**. Indianapolis: New Riders Publishig.

PÉREZ-CASTILLO, Ricardo, et al. **A teaching experience on software reengineering**. 2013 IEEE Global Engineering Education Conference (EDUCON). Berlin, Germany: IEEE, p.1284 - 1293, 2013

PNAE - Programa Nacional de Atividades Espaciais 2005 – 2014. Agência Espacial
Brasileira – **AEB**, 2005. 118p.

PRESSMAN, Roger. **Engenharia de Software: uma abordagem profissional**. 7. ed. Tradução
Ariovaldo Griesi; revisão técnica Reginaldo Arakabi, Julio Arakabi, Renato Manzan de
Andrade. Porto Alegre: AMGH, 2011.

Shin, K.G.; Ramanathan, P. (Jan 1994). "Real-time computing: a new discipline of computer science and engineering" (PDF). *Proceedings of the IEEE*. **82** (1): 6–24. CiteSeerX 10.1.1.252.3947. doi:10.1109/5.259423. ISSN 0018-9219.

SILVEIRA, Paulo; SILVEIRA, Guilherme; LOPES, Sérgio; MOREIRA, Guilherme; STEPAAT, Nico & KUNG, Fabio. **Introdução à Arquitetura e Design de Software**: Uma Introdução à Plataforma Java. São Paulo: Elsevier, 2012.

SOMMERVILLE, Ian. **Engenharia de Software**; tradução de André Maurício de Andrade Ribeiro; revisão técnica de Kechi Hirama. São Paulo: Addison Wesley, 2003.

The Adapter design pattern - Problem, Solution, and Applicability". *w3sDesign.com*. Retrieved 2017-08-12.

The Builder design pattern - Structure and Collaboration". *w3sDesign.com*. Retrieved 2017-08-12.

Interaction Design -
https://www.interaction-design.org/literature/article/what-is-a-usability-engineer

The Strategy design pattern - Problem, Solution, and Applicability". *w3sDesign.com*. Retrieved 2017-08-12.

VIERTOTAK, Mikael. **Launch Rail**. [mensagem pessoal] Mensagem recebida por:
<alexandregarcia@iae.cta.br>. em: 14 abr. 2005.

ANEXO 1: CERTIFICADO DE REGISTRO DE PROGRAMA DE COMPUTADOR: ERUCAE

REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DA ECONOMIA
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS INTEGRADOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512019002129-0**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1° de janeiro subsequente à data de 01/01/2019, em conformidade com o §2°, art. 2° da Lei 9.609, de 19 de Fevereiro de 1998.

Título: ERUCAE - Sistema para cálculo de ajuste do lançador de acordo com os dados do vento

Data de criação: 01/01/2019

Titular(es): UNIVERSIDADE ESTADUAL DO MARANHÃO - UEMA

Autor(es): LUIS CARLOS COSTA FONSECA; JULIO CARDOSO VIDAL DE FIGUEIREDO

Linguagem: JAVA; JAVA SCRIPT

Campo de aplicação: IN-01

Tipo de programa: FA-01; SO-07

Algoritmo hash: SHA-512

Resumo digital hash:
48f3fba0dd21cf6745333ed5304803f12f9229ec03e1301ade93df7bf06de61ba5fbd1b858c03ebf77eceb5fae4fd6fd7a6a00e5bbe447a97154ec22ec3e3729

Expedido em: 01/10/2019

Aprovado por:
Helmar Alvares
Chefe da DIPTO - Portaria/INPI/DIRPA Nº 09, de 01 de julho de 2019