MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

**sid.inpe.br/mtc-m21b/2016/05.13.15.51-TDI**

# PROPOSTA DE UM PROCESSO DE VERIFICAÇÃO POR TESTES BASEADO NA COMPARAÇÃO DAS NORMAS ECSS-E-ST-40C E RTCA-DO-178C E SUA APLICAÇÃO A UM SOFTWARE EMBARCÁVEL

Danilo Gaspar Graça

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Marcelo Lopes de Oliveira e Souza, e Gilberto da Cunha Trivelato, aprovada em 17 de maio de 2016.

URL do documento original:
<http://urlib.net/ 8JMKD3MGP3W34P/3LM86BB>

INPE
São José dos Campos
2016

**PUBLICADO POR:**

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

**sid.inpe.br/mtc-m21b/2016/05.13.15.51-TDI**

# PROPOSTA DE UM PROCESSO DE VERIFICAÇÃO POR TESTES BASEADO NA COMPARAÇÃO DAS NORMAS ECSS-E-ST-40C E RTCA-DO-178C E SUA APLICAÇÃO A UM SOFTWARE EMBARCÁVEL

Danilo Gaspar Graça

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Marcelo Lopes de Oliveira e Souza, e Gilberto da Cunha Trivelato, aprovada em 17 de maio de 2016.

URL do documento original:
<http://urlib.net/ 8JMKD3MGP3W34P/3LM86BB>

INPE
São José dos Campos
2016

Dados Internacionais de Catalogação na Publicação (CIP)

Graça, Danilo Gaspar.

G753p Proposta de um processo de verificação por testes baseado na comparação das normas ECSS-E-ST-40C E RTCA-DO-178C e sua aplicação a um software embarcável / Danilo Gaspar Graça. – São José dos Campos : INPE, 2016.

xxiv + 155 p. ; (sid.inpe.br/mtc-m21b/2016/05.13.15.51-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2016.

Orientadores : Drs. Marcelo Lopes de Oliveira e Souza, e Gilberto da Cunha Trivelato.

1. Testes de software. 2. Verificação. 3. Processos de desenvolvimento. 4. Desenvolvimento de software. 5. Sistemas aeroespaciais. I.Título.

CDU 629.7:004.4

Aluno (a): ***Danilo Gaspar Graça***

**Título:** "PROPOSTA DE UM PROCESSO DE VERIFICAÇÃO POR TESTES BASEADO NA COMPARAÇÃO DAS NORMAS ECSS-E-ST-40C E RTCA-DO-178C E SUA APLICAÇÃO A UM SOFTWARE EMBARCÁVEL **".**

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de ***Mestre*** em

***Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais***

Dra. Ana Maria Ambrosio

***Presidente / INPE / São José dos Campos - SP***

Dr. Marcelo Lopes de Oliveira e Souza

***Orientador(a) / INPE / SJCampos - SP***

Dr. Gilberto da Cunha Trivelato

***Orientador(a) / HOMINE / São José dos Campos - SP***

Dr. Guilherme Conceição Rocha

***Convidado(a) / Konatus / São José dos Campos - SP***

***Este trabalho foi aprovado por:***

***( ) maioria simples***

***(X) unanimidade***

***São José dos Campos, 17 de Maio de 2016***

*“Você é livre para fazer suas escolhas, mas é prisioneiro das consequências. ”*

*Pablo Neruda*

*A minha mãe; à tia Letícia e a meus irmãos.*

## AGRADECIMENTOS

Agradeço ao Dr. Gilberto da Cunha Trivelato pela sugestão do tema de estudo, pela orientação, por humildemente compartilhar seus conhecimentos e pela convivência dos últimos anos.

Agradeço ao professor Dr. Marcelo Lopes de Oliveira e Souza pela orientação, por disponibilizar seu precioso tempo e pelo empenho mantido na busca por um ensino de altíssimo nível.

Também, agradeço à LDRA que forneceu as ferramentas utilizadas no trabalho, através da empresa Konatus e seu diretor Dr. Guilherme Conceição Rocha e também ao Diogo Pereira e Philipe Massad pelas horas de auxílio e suporte na ferramenta.

Agradeço à empresa Homine Informática Educação e Tecnologia Ltda. pela oportunidade de desenvolvimento deste mestrado, no seu programa de formação, como parte das minhas atividades de trabalho, fornecendo equipamentos, remuneração e benefícios associados.

Agradeço ao INPE, aos professores, aos colegas do Curso de Engenharia e Tecnologia Espaciais na área de Engenharia e Gerenciamento de Sistemas Espaciais e, em espacial, aos colegas do LabSystems, Marize Simões e Luiz Lencioni.

## RESUMO

Este trabalho apresenta uma proposta de um processo de verificação por testes baseado na comparação das normas ECSS-E-ST-40C e RTCA-DO-178C e sua aplicação a um software espacial embarcável. Isto inclui: a revisão bibliográfica sobre conceitos de Engenharia de Software e sobre as normas abordadas das indústrias espacial e aeronáutica; a comparação e análise da estrutura de processos das referidas normas para desenvolvimento de software; o estudo de ferramentas semiautomáticas de testes; a proposta de métodos e procedimentos básicos de um processo de verificação por testes baseado na comparação das normas ECSS-E-ST-40C e RTCA-DO-178C com suas principais atividades para ser aplicado no desenvolvimento de softwares espaciais embarcados; e a validação do processo proposto através de sua aplicação a um estudo de caso. A aplicação do processo proposto mostrou que i) a inversão da sequência tradicional de testes apresenta significativos ganhos em termos de esforço; ii) ela só é possível com a utilização de uma ferramenta de testes que tem impacto direto no processo; iii) a maior contribuição da ferramenta é compartilhar os resultados da aplicação de testes em um nível com os demais níveis (HW/SW, SW/SW e baixo nível); iv) os métodos e procedimentos utilizados reduzem significativamente o esforço aplicado na realização das atividades de teste.



Palavras-chave: Testes de software. Verificação. Processos de desenvolvimento. Desenvolvimento de software. Sistemas aeroespaciais.

**PROPOSAL FOR VERIFICATION PROCESS BY TESTING BASED ON COMPARISON OF THE ECSS-E-ST-40C AND RTCA-DO-178C STANDARDS AND ITS APPLICATION TO EMBEDDABLE SOFTWARE**

## ABSTRACT

This work presents a proposal for verification process by testing based on the comparison of ECSS-E-ST-40C and RTCA-DO-178C standards and its application to embeddable software. This includes: a bibliographic review on software engineering concepts and the space and aeronautics industries standards used; a comparison and analysis of the structure of processes of the mentioned standards for software development; a study of semiautomatic tools for tests; a proposal of basic methods and procedures of a verification process by testing based on comparison of ECSS-E-ST-40C and RTCA-DO-178C standards with its main activities to be applied in the development of space embedded; the validation of the proposed process through its application to case study. The application of the proposed method showed i) reversing the traditional test sequence shows significant gains in terms of effort; ii) it is possible only with the use of a test tool has a direct impact on process; iii) the greatest tool contribution is to share the results of the application tests in one level with other levels (HW/SW, SW/SW and Low-Level); iv) the methods and procedures used significantly reduce the effort applied in carrying out the testing activities.

Keywords: Software Testing. Verification. Development Process. Software Development. Aerospace Systems.

## LISTA DE FIGURAS

**Pág**.

## LISTA DE TABELAS

**Pág**.

## LISTA DE SIGLAS E ABREVIATURAS

| | |
|---|---|
| ANAC | Agência Nacional de Aviação Civil |
| ARP | Aerospace Recommended Practice |
| CBERS | China-Brazil Earth-Resources Satellite |
| CTA | Centro Técnico Aeroespacial |
| DAL | Development Assurance Level |
| DC | Decision Coverage |
| DD | Dependence Diagram |
| DEA | Divisão de Eletrônica Aeroespacial |
| DO | Design Ordnance |
| DoDAF | Department of Defense Architecture Framework |
| ECSS | European Cooperation on Space Standardization |
| EASA | European Aviation Safety Agency |
| ESA | European Space Agency |
| ETE | Engenharia e Tecnologia Espaciais |
| FAA | Federal Aviation Administration |
| FDAL | Function Development Assurance Level |
| FHA | Functional Hazard Assessment |
| FINEP | Financiadora de Estudos e Projetos |
| FMEA | Failure Mode and Effects Analysis |
| FMES | Failure Modes and Effects Summary |
| FTA | Fault Tree Analysis |
| IAR | Ingenjörsfirman Anders Rundgren |
| IBIT | Initiated Built-In Test |
| IDAL | Item Development Assurance Level |
| IDE | Integrated Development Environment |
| INCOSE | International Council on Systems Engineering |
| INPE | Instituto Nacional de Pesquisas Espaciais |
| MA | Markov Analysis |

| | |
|---|---|
| MC | Modified Condition |
| MDE | Model-Based Engineering |
| MECB | Missão Espacial Completa Brasileira |
| NASA | National Aeronautics and Space Administration |
| PDR | Preliminary Design Review |
| PNAE | Programa Nacional de Atividades Espaciais |
| PSAC | Plan for Software Aspects of Certification |
| PSSA | Preliminary System Safety Assessment |
| RTCA | Radio Technical Commission for Aeronautics |
| SAE | Society of Automotive Engineers |
| SCI | Software Configuration Index |
| SCM | Software Configuration Management |
| SDP | Software Development Plan |
| SECI | Software Life Cycle Environment Configuration Index |
| SIA | Sistemas Inerciais para Aplicação Aeroespacial |
| SISCAO | Sistema de Controle de Atitude e Órbita |
| SISNAVLS | Sistema Inercial de Navegação para Veículo Lançador de Satélites |
| SQA | Software Quality Assurance |
| SRR | Software Requirements Review |
| SSA | System Safety Assessment |
| SUBORD | Supervisão de Bordo |
| SVP | Software Verification Plan |
| SWRR | Software Requirements Review |
| SYSML | Systems Modeling Language |
| TQL | Tool Qualification Level |
| UML | Unified Modeling Language |

## SUMÁRIO

# 1 INTRODUÇÃO

## 1.1. Contexto

A capacidade das organizações, dos produtos e dos serviços em competir, adaptar e até sobreviver depende e dependerão cada vez mais de software, especialmente embarcados.

No setor aeroespacial, os sistemas estão cada vez mais complexos e/ou altamente integrados devido ao número de funcionalidades entregues e a gama de aplicações existentes que fazem uso de software crítico. Isto inclui: controle de veículos aeroespaciais como satélites, foguetes, aeronaves e gerenciamento de sensores, telemetria e telecomando, etc. O desenvolvimento de aplicações aeroespaciais embarcadas requer um rigoroso processo de desenvolvimento, em especial, de verificação e validação. Esse deve atender aos padrões de segurança considerando o nível de criticalidade inerente guiado por normas como a ECSS-E-ST-40C e RTCA-DO-178C.

O setor espacial conta com várias normas para o desenvolvimento de sistemas e software. Dentre elas, destacam-se as normas da ECSS (*European Cooperation on Space Standardization*) com aplicações nos campos de Gerenciamento, Engenharia e Qualidade de produto. Em particular, a norma ECSS-E-ST-40C define os princípios e requisitos necessários ao desenvolvimento de softwares espaciais.

O INPE (Instituto Nacional de Pesquisas Espaciais), referência no setor espacial no Brasil, utiliza como referência os processos descritos nas normas ECSS, em especial, a norma E-ST-40C para o desenvolvimento de software. O INPE customiza os processos para atender à realidade de cada projeto, como pode ser observado nos projetos CBERS3 e CBERS4 (ALBUQUERQUE; PERONDI, 2010; YASSUDA, 2010). Desde sua fundação, o INPE vem desenvolvendo sua metodologia e processo, tendo como referência inicial sua experiência no desenvolvimento da Missão Espacial Completa Brasileira

(MECB), obtida em treinamentos contratados para auxiliar na montagem e condução de um programa espacial. Devido a isso, ao longo dos anos, os processos foram se moldando ao padrão adotado pela ESA - Agência Espacial Europeia (YASSUDA, 2013).

O setor aeronáutico conta com várias normas para o desenvolvimento de sistemas e software. Dentre elas, destacam-se as normas da RTCA (*Radio Technical Committee for Aeronautics*) com aplicações nos campos de Gerenciamento, Engenharia e Qualidade de produto. Em particular, a norma DO-178C provê um guia para o desenvolvimento de software para sistemas e equipamentos aeronáuticos. Sucedendo a DO-178B, a nova versão dispõe de quatro suplementos: *Software Tool Qualification Considerations* (DO-330), *Model Based Development and Verification* (DO-331), *Object-Oriented Technology and Related Techniques* (DO-332), *Formal Methods Supplement* (DO-333). Esta nova versão apresenta mais objetivos para os níveis A, B e C, em uma linguagem mais clara e terminologia atualizada, auxiliando em sua consistência.

Com o aumento da complexidade dos processos de software e o surgimento de poderosas ferramentas, sugere-se adotar ambientes que suportam e automatizem atividades como gestão de requisitos, gestão de configuração e, sobretudo, as atividades de testes. Nesse contexto (REIS; AMBROSIO; FERREIRA, 2012) afirmam que o gerenciamento dos testes gera uma grande quantidade de informações que são aproveitadas de maneira mais eficiente quando manipuladas através de uma ferramenta especialista.

Portanto, devido ao rigor das normas supracitadas e seu impacto no desenvolvimento de sistemas, as atividades de verificação, principalmente por testes, são importantes aliadas na busca por construir produtos com qualidade e confiabilidade.

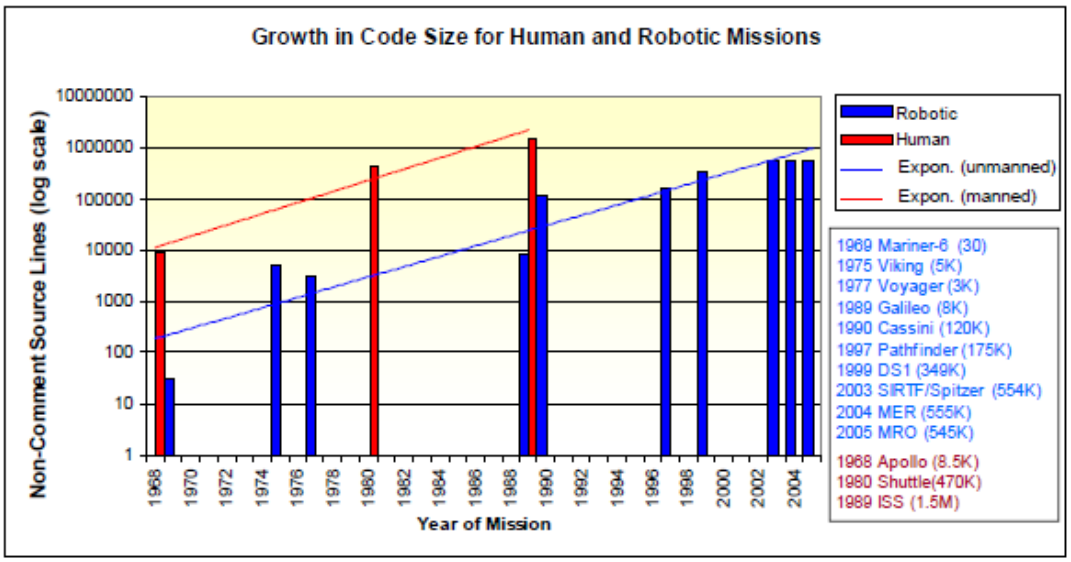
## 1.2. Objetivo Do Trabalho

O objetivo do trabalho é apresentar uma proposta de um processo de verificação por testes baseado na comparação das normas ECSS-E-ST-40C e RTCA-DO-178C e sua aplicação a um software embarcável. Para isto, foram adotados os passos descritos na Metodologia.

## 1.3. Motivação e Justificativa

O aumento da complexidade dos sistemas aeroespaciais, aliado ao rigoroso processo de desenvolvimento e ao crescimento da quantidade de software embarcados nestes sistemas, como exemplificado pela Figura 1.1, sugere a adoção de normas também como referência no processo de desenvolvimento de softwares aeroespaciais. O histórico das normas exige a maturidade do processo de desenvolvimento e gera referências rigorosas para o desenvolvimento de softwares críticos.

Figura 1.1 – Histórico do crescimento de software em missões espaciais




Fonte: NASA Study on Flight Software Complexity (2009).

A disponibilidade e o surgimento de novas e modernas ferramentas auxiliam o desenvolvimento de software, a definição e manutenção de processos em todas as fases do desenvolvimento, assim contribuindo para a produção de

software com qualidade. Já no contexto de validação, representar casos de teste através de ferramentas possibilita a criação de testes mais elaborados e complexos, especialmente no caso de softwares críticos (BERNARDO; KHON, 2008).

Como já mencionado, devido à complexidade dos sistemas e à vasta gama de aplicações que usam softwares, as áreas de desenvolvimento precisam de sistemas de qualidade certificada, pois o funcionamento deles é crítico para o sucesso da missão ou objetivo para qual o software foi implementado. Em função da área de atuação e da classificação de severidade do software, além do desenvolvimento utilizar uma norma, o produto final contendo este software deve ser certificado por um órgão, como, por exemplo, FAA (*Federal Aviation Administration*) para o setor aeronáutico.

Como exemplo, na área aeronáutica, a norma DO-178C consequente revisão da DO-178B, classifica as condições de falha em cinco níveis de severidade (catastrófica, perigosa, maior, menor e sem efeito) que impõe níveis de rigor no processo de desenvolvimento do software associado. Conforme apontado por SQS (2012), ela é considerada um sucesso na história da indústria aeronáutica, pois durante os mais de 20 anos de validade da DO-178B e apesar do aumento da complexidade dos sistemas, houve uma redução no número de acidentes relacionados a falhas com origem em erros de software. Na área espacial, as normas da ECSS e, especificamente, a E-ST-40C, que trata do desenvolvimento de software, representa uma coleção de boas práticas das indústrias e agências europeias e vem sendo utilizada como uma boa referência no setor.

Especialmente em projetos espaciais, erros de software que culminaram em falhas geraram grandes prejuízos, como exemplificam os incidentes apresentados na Tabela 1.1.

Dois acontecimentos marcantes também demonstram tais prejuízos:

- Falha ocorrida durante o lançamento do Ariane 5, que explodiu por causa de uma falha que teve como origem um problema de software, gerou prejuízos da ordem de 300 milhões de dólares (HECHT, et al., 2005).

- Falha de transmissão de um dos canais de comunicação da Sonda Huygens da NASA prejudicou o envio das imagens, que passaram a ser transmitidas pela metade (DEUTSCH, 2002).

Tabela 1.1 – Histórico de falhas que culminaram em perdas.

| Projeto | Incidente | Origem da Falha |
|---|---|---|
| Airbus A320 **Junho 1988** | Caiu ao fazer um voo de apresentação. | Limitação de operação por definição incorreta de variável de software. |
| ARIANE 501 **Junho 1996** | Saiu da trajetória, quebrou-se e explodiu. | Erro de conversão no software do sistema de referência inercial. |
| Mars Climate Orbiter **Setembro 1998** | Perda do Satélite. | Troca de unidades de medida. |
| Mars Pathfinder **Dezembro 1998** | Perda do Satélite. | Erro de software forçou desligamento prematuro da máquina de pouso. |
| Titan IV B **Abril 1999** | Falha ao colocar o satélite na sua órbita final. | Programação incorreta do computador de orientação que causou perda do controle de atitude. |
| Delta-3 **Abril 1999** | Lançamento abortado. | Falha do software de bordo na inicialização da máquina principal. |
| PAS-9 **Março 2000** | Perda do satélite de comunicação ICO F-1. | Erro na atualização do software de solo. |

Fonte: Adaptado de Ambrósio (2005) e Mazza (2000).

Os acidentes demonstram que pequenos incidentes com erros de software podem ocasionar severas falhas com significativas perdas financeiras. Portanto, para atestar o correto funcionamento ou a presença de erros, atividades de qualidade, verificação e validação devem ser realizadas ao longo do desenvolvimento do software.

Segundo Reis, Ambrósio e Ferreira (2012), uma das atividades do ciclo de desenvolvimento de um software que aumenta a qualidade final é a realização de testes.

Com o processo de verificação por testes que propomos espera-se ter um conjunto de atividades básicas de teste, e ainda através da inversão da tradicional sequência de testes e uso de uma ferramenta seja possível melhorar a gestão e execução dos procedimentos.

Considerando que o uso destas normas é uma tendência mundial, o exercício da sua aplicação conjunta para o desenvolvimento de softwares de sistemas espaciais é oportuno e possibilita contribuir para a redução de falhas causadas por erros nesse tipo de software.

### 1.4. Metodologia

Para a realização do trabalho desta dissertação, adotamos os seguintes passos:

- Levantamento bibliográfico;
- Estudo das normas de desenvolvimento de software das indústrias espacial e aeronáutica, especialmente ECSS-E-ST-40C e RTCA-DO-178C;
- Análise e comparação da estrutura dos processos das normas anteriormente citadas;
- Definição de um processo de verificação por testes simplificado baseado na comparação das normas de desenvolvimento de software ECSS-E-ST-40C e RTCA-DO-178C;
- Estudo sobre o uso de ferramentas de testes semiautomáticas no processo definido.

## 1.5. Organização Do Trabalho

No Capítulo 1, tem-se a introdução com contexto, objetivos do trabalho, metodologia, motivação e justificativa e a organização do trabalho. O Capítulo 2 é dedicado a uma exposição dos conceitos básicos e revisão bibliográfica. No Capítulo 3 apresentamos a comparação entre as normas de desenvolvimento de software da indústria espacial e da indústria aeronáutica, respectivamente, ECSS-E-ST-40C e RTCA-DO-178C. No Capítulo 4 é apresentada uma proposta de processo de verificação por testes baseado na comparação das normas ECSS-E-ST-40C e RTCA-DO-178C. No Capítulo 5 teremos a aplicação do processo proposto através de um estudo de caso da indústria. Finalmente, no Capítulo 6, são apresentadas as conclusões, recomendações e sugestões de trabalhos futuros.

# 2 CONCEITOS BÁSICOS E REVISÃO DA LITERATURA

Este capítulo dedica-se a apresentar conceitos e normas que são abordadas neste documento e também objetos de estudo, com o intuito de embasar o entendimento do trabalho proposto.

## 2.1. Modelos de Processos de Software

Relevantes aspectos devem ser considerados no processo de desenvolvimento de software. Este é constituído por fases que são planejadas de acordo com o modelo de processo de software adotado. Em geral, os modelos possuem ao menos quatro fases: especificação, projeto, desenvolvimento e verificação.

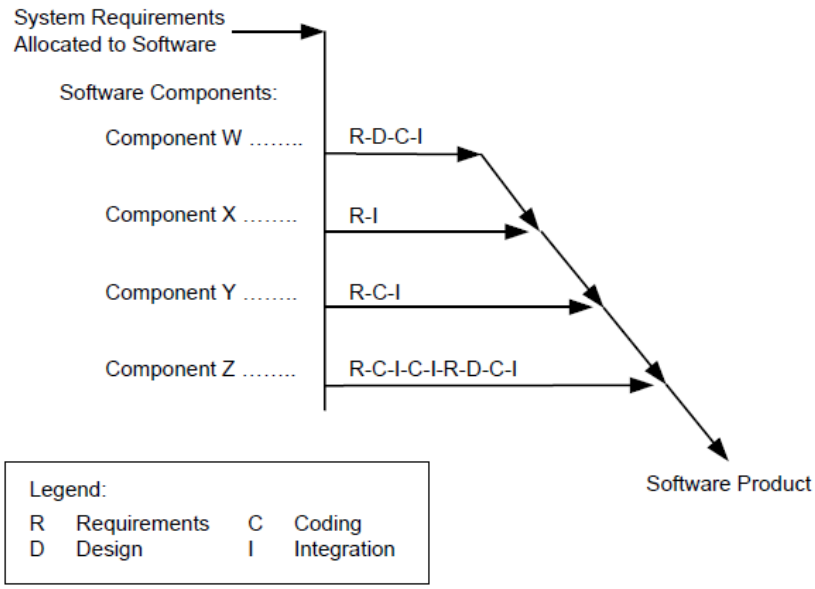
Portanto, ao se definir um processo é necessário definir qual o modelo de processo utilizado, pois características como fases, macro atividades, dependência e precedência de tarefas, métodos e artefatos são oriundos desse modelo. Esse, por sua vez, reflete o software a ser desenvolvido quanto à estabilidade dos requisitos, complexidade, uso de um componente de software previamente desenvolvido, estratégia de desenvolvimento, etc.

Ao longo do tempo, o modelo de processo ou modelo de ciclo de vida clássico conhecido como **modelo em cascata**, sofreu alterações para acomodar novas técnicas e, principalmente, para reparar desvantagens, dando origem a outros modelos, como **espiral**, **iterativo**, **V** e **incremental**.

A norma RTCA-DO-178C aborda a possibilidade do desenvolvimento de componentes de software a partir de modelos de ciclos de vida diferentes, como ilustra a Figura 2.1, embora o processo de integração seja incremental e alinhado com as atividades do processo de verificação.

Quanto à norma E-ST-40C, os processos apresentam uma estrutura linear (ver Figura 2.3 na seção 2.6.2) próximo ao modelo em cascata, indicando o uso de uma abordagem mais tradicional.

Figura 2.1 – Exemplos de diferentes sequências de desenvolvimento de software.




Fonte: Adaptado de RTCA-DO-178C (2012).

Contudo, a adoção de uma abordagem expressa o foco na produtividade e qualidade, adicionalmente aliada à outra atividade como gerenciamento do projeto, tornando possível o desenvolvimento do software conforme metas de custos, prazos e, sobretudo, qualidade.

## 2.2. Criticalidade

**Software crítico** é aquele que contém funções de software críticas em relação à segurança. O desenvolvimento de software crítico deve, necessariamente, levar em consideração todos os estados perigosos do sistema que sejam alcançáveis devido à atuação do software. Eles devem ser rastreados desde a fase de levantamento de requisitos até a fase de aceitação do sistema (ALONSO, 1998).

Logo, considera-se software crítico, aquele cujo erro leva a uma falha (*fault*) da função que pode causar uma falha do sistema (*failure condition*) que possa resultar na perda de vidas humanas e danos ao meio ambiente. Áreas como nuclear, aeroespacial, ferroviária, automobilística e médica, buscam sistemas com certificação de qualidade, onde o reconhecimento é alcançado através do

cumprimento de rigorosas normas de desenvolvimento que asseguram que aquele software crítico é “seguro”.

Ainda segundo Peña e Souza (2013), os produtos de software seguros e críticos podem ser classificados dependendo da chamada condição de falha, isto é, em função da severidade do efeito da falha.

Tabela 2.1 – Classificação de severidade das consequências para área espacial.

| **Severidade** | **Nível** | **Confiabilidade (ECSS-Q-ST-30)** | **Segurança (ECSS-Q-ST-40)** |
|---|---|---|---|
| **Catastrófica (*Catastrophic*)** | 1 | Propagação da Falha | • Perda de vida, risco de vida, lesão ou doença incapacitante permanente.<br>• Perda de sistema.<br>• Perda de uma interface do sistema de voo tripulado.<br>• Perda de instalações do local de lançamento.<br>• Severos efeitos prejudiciais ambientais. |
| **Crítica (*Critical*)** | 2 | Perda de Missão | • Afastado temporariamente, mas sem lesões com risco à vida ou doença ocupacional;<br>• Grandes danos à interface do sistema de voo;<br>• Grandes danos a instalações terrestres;<br>• Danos à propriedade pública ou privada;<br>• Os principais efeitos ambientais prejudiciais. |
| **Grande (*Major*)** | 3 | Grande Degradação de Missão | --- |
| **Menor ou desprezível (*Minor or Negligible*)** | 4 | Pequena Degradação de Missão ou sem efeito. | --- |

Fonte: ECSS-Q-ST-30C (2009).

Cada conjunto de normas possui métodos para avaliação e classificação próprios para categorizar a severidade dos impactos no sistema a partir de eventos indesejados. A Tabela 2.1 apresenta a classificação da severidade das consequências das falhas conforme a norma Q-ST-30C que trata de Dependabilidade (*Dependability*), base para definir as categorias de severidade da função de software, as quais descritas e detalhadas mais à frente. Ainda de acordo com esta tabela e a norma Q-ST-40C, os níveis de severidade **Grande** (*Major*) e **Menor ou Desprezível** (*Minor or Negligible*) não apresentam descrição sobre aspectos relacionados com a segurança.

A avaliação de dependabilidade e segurança são conduzidas, respectivamente, de acordo com os requisitos das normas Q-ST-30C e Q-ST-40C, dentro da fase de desenvolvimento e alocação dos requisitos de sistemas.

A Tabela 2.2 mostra a classificação e descrição dos cinco níveis de condições de falhas reconhecidas pela norma DO-178C, as quais são utilizadas na categorização de severidade destas condições de falhas e nível de software, isto com as considerações de normas que orientam certificação e o desenvolvimento de sistemas.

Tabela 2.2 – Classificação das condições de falha no setor aeronáutico.

| **Condição de falha** | **Descrição** |
|---|---|
| **Catastrófica (*Catastrophic*)** | Uma condição de falha que causa mortes ou perda do veículo ou arma. |
| **Crítica (*Hazardous*)** | Uma condição de falha que pode causar grandes danos humanos severos ou danificação do veículo ou sistema que causa perda da missão. |
| **Maior (*Major*)** | Uma condição de falha que causa danos menores e pode provocar atraso ou degradação da missão. |
| **Menor (*Minor*)** | Uma condição de falha que não provoca danos, mas deve ser solucionada. |
| **Sem Efeitos (*No Safety Effect*)** | Sem efeitos na segurança. |

Fonte: RTCA-DO-178C (2012).

## 2.3. Validação

O termo **validação** é encontrado e utilizado segundo muitos significados, sendo que, muitas das vezes, erroneamente. Buscamos apresentar a seguir como algumas referências importantes fazem uso do termo e, na sequência, iremos propor uma definição pela qual iremos nos basear.

Segundo Sommerville (2011), o objetivo da validação é garantir que o software atenda às expectativas do cliente. A validação é essencial porque as especificações nem sempre refletem os desejos ou necessidades dos clientes e usuários do software.

Segundo o INCOSE (2006), a validação do produto confirma que o produto construído satisfaz as necessidades do *stakeholder*, garantindo que os requisitos e a implementação provêem a solução correta para o problema do cliente.

Segundo a norma E-ST-40C, a validação é o processo que demonstra que o produto está de acordo para realizar seu uso pretendido e no ambiente pretendido.

A norma DO-178C apresenta uma definição para o termo validação, mas limitado ao escopo de requisitos.

Baseado em todos, definimos como validação do produto o processo de avaliar se o produto satisfaz às expectativas do usuário conforme um documento de requisitos através de métodos como análise, revisão, inspeção e teste. Avalia a adequação jurídica. A validação deve vir de encontro às necessidades dos *stakeholders* tratados.

## 2.4. Verificação

Do mesmo modo, o termo **verificação** pode variar conforme o contexto, autores e literaturas. Deste modo, buscaremos aqui uma definição para nos orientarmos neste trabalho.

Segundo Lake (1999), verificação do sistema é o processo de avaliar um sistema para determinar se o produto ao final de uma fase de desenvolvimento satisfaz a condição imposta no início desta fase.

Segundo a norma E-ST-40C, verificação do produto é o processo que demonstra através de evidências objetivas que um produto é projetado e produzido de acordo com suas especificações.

Segundo a norma DO-178C (2012), verificação é a avaliação da saída de um processo para garantir a adequação no que diz respeito às entradas e normas previstas para esse processo.

Baseado em todos, definimos como verificação do produto o processo de avaliar se o produto satisfaz às boas práticas conforme uma base de conhecimento, padrão ou norma, das áreas afins através de métodos como análise, revisão, inspeção e teste. É um ato de correção técnica. A verificação deve vir ao encontro das verdades dos conhecimentos tratados.

## 2.5. Testes

Teste de software é um método de verificação do código fonte, no qual procuramos erros através da execução do software sob determinada condição.

Conforme aborda Myers et al (2012), testes de software é um processo, ou uma série de processos, projetados para deixar claro que o software faz o que o cliente deseja e não faz o indesejado. Nele a execução de um programa é realizada com o intuito de encontrar erros.

Segundo DELAMARO et al. (2007), teste de software é uma atividade dinâmica, e se baseia na execução de um programa ou de um modelo, utilizando algumas entradas em particular e verificando se seu comportamento está de acordo com o esperado.

Segundo a norma DO-178C, teste é uma das atividades do processo de verificação de software, juntamente com uma combinação de revisões e análises.

Testes de software possuem um extenso conjunto de tipos, técnicas e abordagens. Neste trabalho, faremos uso basicamente de dois tipos: testes do tipo caixa preta e caixa branca.

Na abordagem de teste do tipo **caixa preta**, o software é tratado do ponto de vista de entradas e saídas, o código fonte do software não é conhecido. Esta abordagem também é conhecida como **funcional**, dado que o procedimento para definir os casos de teste tem por base o mapa de funções do componente ou sistema, e isto sem fazer qualquer referência à estrutura interna destes.

Algumas das técnicas utilizadas para os testes caixa preta ou funcionais são:

- **Classe de equivalência:** os casos de teste são desenvolvidos com base em intervalos de valores, os quais possuem o mesmo comportamento na execução do software. Tais intervalos determinam o que é chamado de classe de equivalência.

- **Valores limite (*Limit Values*):** os casos de testes são definidos conforme os valores máximos e mínimos dos intervalos, uma vez que o limite das classes é mais propenso a esconder diferentes tipos de erros. O uso de valores limites reduz o tamanho do conjunto de testes a ser utilizado.

- **Valores inválidos (*Invalid Values*):** os casos de teste são definidos com base em valores inválidos para verificar o comportamento do software. Também mencionado em algumas literaturas como **teste de estresse ou robustez**.

A abordagem do teste do tipo **caixa branca** é inversa ao tipo caixa preta, pois nesta abordagem a estrutura e implementação do software é considerada.

Também conhecido como testes **não funcionais**, este tipo avalia como o software foi desenvolvido, o que requer um avançado conhecimento do código. Algumas técnicas são utilizadas para avaliar a estrutura de código, variando quanto à profundidade e detalhamento.

Uma das técnicas utilizadas é a análise da cobertura de código, a qual avalia quais partes da estrutura do código foram exercitadas a partir da execução dos testes baseados em requisitos. Com impacto na qualidade do produto de software, a cobertura de código também se apresenta como um objetivo de verificação importante diante de uma certificação, neste trabalho aborda-se três tipos, a saber:

- **Cobertura de declaração (*Statement Test*):** avalia quais linhas executáveis de código foram executadas. Este método não considera laços de repetição e decisões condicionais. É considerado uma boa métrica, mas possui limitações, como não avaliar todas as condições de um ramo.

- **Cobertura de Ramo (*Branch Test*):** avalia ramos e decisões para todas as saídas. Possui limitações ao trabalhar com linguagens que permitam o *Lazy Evaluation* (técnica que atrasa a computação de uma parte do código).

- **Cobertura de Decisão/Condição Modificada (*Modified Condition/Decision Coverage - MC/DC*):** Este é um tipo mais completo de análise de cobertura, pois avalia e reporta a saída de ramos condicionais complexos, avaliando se todas as combinações foram testadas. Cobre as desvantagens dos tipos anteriores, como *lazy evaluation.*

Ainda com relação aos testes não funcionais, pode-se avaliar com base na estrutura do código fonte três métricas diretamente relacionadas à qualidade do produto de software, as quais são compostas por medidas retiradas através da

análise deste código. A seguir apresentam-se essas métricas e suas respectivas medidas, além do limite superior e inferior apresentado para cada métrica.

- **Clareza (*Clarity*):** a clareza do código fonte tem como característica a facilidade de entendimento de seu funcionamento, sendo este diretamente associado ao estilo de programação. A Tabela 2.3 presenta as medidas utilizadas para avaliar quanto a esta métrica.

Tabela 2.3 – Métrica de clareza do código.

| Métrica | Descrição | Limite Inferior | Limite Superior |
|---|---|---|---|
| Profundidade de aninhamento de laços de repetição *(Depth of loop nesting)* | Quantidade de laços de repetição aninhados. | 0 | 2 |
| Tamanho médio de blocos básicos *(Average length of basic blocks)* | Média do comprimento dos blocos básicos de código. | 1% | 6% |
| Comentários de código / Linhas executáveis *(Code comments / exec. lines)* | Relação de linhas de comentários de código por linhas executáveis. | 5 | 200 |
| Comentários de declaração / Linhas executáveis *(Declaration comments / exec. Lines)* | Relação de linhas de comentários em declaração por linhas executáveis. | 1 | 100 |
| Total de Comentários / Linhas executáveis *(Total comments / exec. Lines)* | Relação do total de linhas de comentários por linhas executáveis. | 10 | 200 |
| Linhas em Branco *(Blank lines)* | Quantidade de linhas em branco. | 0 | 100 |
| Comentários em código executável (Comments *in executable code)* | Quantidade de comentários em código executável. | 1 | 100 |
| Comentários em declaração *(Comments in declarations)* | Quantidade de comentários em declarações. | 0 | 100 |
| Comentários em cabeçalhos *(Comments in headers)* | Quantidade de comentários em cabeçalhos. | 5 | 50 |
| Total de comentários *(Total comments)* | Quantidade total de comentários. | 10 | 200 |

Fonte: Produção do autor.

- **Manutenabilidade (*Maintenability*):** como apontado por Sommerville (2011), a manutenção de software é a atividade de efetuar mudanças controladas visando correção, adaptação ou aumento de um software, onde se pode definir a manutenabilidade de código fonte como a facilidade de aplicar estas alterações. A Tabela 2.4 apresenta as métricas relacionadas com a estrutura do código e que impactam diretamente sua manutenção.

Tabela 2.4 – Métricas de manutenabilidade do código.

| Métrica | Descrição | Limite Inferior | Limite Superior |
|---|---|---|---|
| Complexidade ciclomática *(Cyclomatic complexity)* | Quantidade de caminhos linearmente independentes. | 1 | 10 |
| Nós *(Knots)* | Quantidade de nós, o qual sendo o encontro de dois ou mais caminhos. | 0 | 5 |
| Complexidade ciclomática essencial *(Essential cyclomatic complexity)* | Grau que um módulo de software possui construções desestruturadas. | 1 | 3 |

Fonte: Produção do autor.

- **Testabilidade (*Testability*):** a verificação por testes visa assegurar qualidade e gerar evidências de um produto de software que atenda as especificações funcionais com o mínimo de defeito, neste contexto a testabilidade pode-se definir como a capacidade do código fonte de expor estes defeitos de acordo com as características presentes no código, as quais são inseridas no desenvolvimento. A Tabela 2.5 apresenta estas características ou métricas relacionadas com a testabilidade de código.

Tabela 2.5 – Métricas de testabilidade do código.

| Métrica | Descrição | Limite Inferior | Limite Superior |
|---|---|---|---|
| Espalhamento de saída *(Fan-out)* | Quantidade de objetos ou módulos que dependem de um método direta ou indiretamente. | 0 | 5 |
| Espalhamento de entrada *(File fan-in)* | Quantidade de objetos ou módulos que o método depende. | 0 | 5 |
| Pontos de Saída de procedimentos *(Procedure exit points)* | Quantidade de pontos de saída de um método. | 0 | 1 |
| Número de laços de repetição *(Number of loops)* | Quantidade de laços de repetição presentes no código. | 0 | 4 |
| Número de blocos básicos *(Number of basic blocks)* | Quantidade de blocos básicos de código. | 1 | 30 |
| Complexidade ciclomática *(Cyclomatic complexity)* | Quantidade de caminhos linearmente independentes. | 1 | 10 |
| Nós *(Knots)* | Quantidade de nós, representado pelo encontro de dois ou mais caminhos. | 0 | 5 |

Fonte: Produção do autor.

A adoção dos parâmetros para limite superior e limite inferior utilizados nas Tabela 2.3, Tabela 2.4 e Tabela 2.5, justifica-se com o uso dos parâmetros utilizados e retirados da ferramenta LDRA TBrun, a qual será detalhada mais adiante na seção 5.2 (Detalhando a Ferramenta), uma vez que definido estas métricas de software não se encontra no escopo deste trabalho.

Contundo, algumas literaturas e fontes também consideram os testes caixa preta ou testes funcionais como **análise dinâmica**, enquanto os testes caixa branca ou testes não funcionais como **análise estática**.

## 2.6. Normas da Indústria Espacial

Devido ao INPE utilizar, historicamente, no desenvolvimento de programas espaciais e métodos e processos derivarem do conjunto de normas ECSS, este trabalho usará como referência algumas normas desse conjunto. Um estudo bastante completo e atualizado sobre as normas espaciais, aeronáuticas e

industriais realizado no INPE é apresentado por Junior (2013). As normas que farão parte do escopo do trabalho serão apresentadas nas seções seguintes.

### 2.6.1. ECSS-E-ST-10C

A norma ECSS-E-ST-10C intitulada "*System Engineering General Requirements*" (Requisitos Gerais para Engenharia de Sistemas) define uma implementação da engenharia de sistemas para o desenvolvimento de produtos e sistemas espaciais.

Esta norma detalha as fronteiras da Engenharia de Sistemas, especificando tarefas, seus objetivos, entradas e saídas em cada uma das fases de projeto. Tem também como um de seus objetivos implementar o modelo cliente-fornecedor *"customer-system-supplier model"*, no qual a decomposição de um sistema define o cliente como o responsável por um produto e o fornecedor com o responsável pelo desenvolvimento de um subproduto. Esse mesmo modelo é utilizado pelas demais normas do conjunto ECSS.

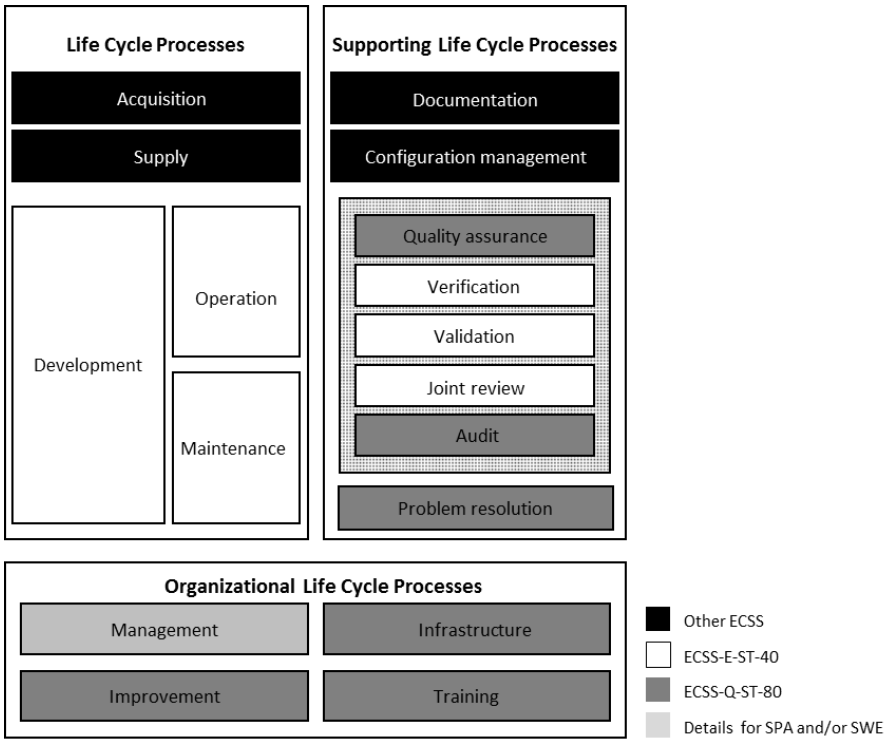
Segundo tal norma, requisitos específicos relacionados com a Engenharia de Sistemas, como verificação e teste são tratados em normas do conjunto ECSS como "E-ST-10-02 *Space Engineering – Verification*"; e disciplinas ou elementos específicos são tratadas em normas dedicadas do conjunto ECSS, como "M-ST-40 *Space project management – Project planning and implementation*".

Entretanto, com relação a isso, em alguns casos não há informações de relacionamento com outras normas, por exemplo, como observado por Reginato (2012): ele menciona que, no escopo da norma E-ST-10C, não é encontrada referência para outra norma que inclua as análises de confiabilidade.

### 2.6.2. ECSS-E-ST-40C

Esta norma de Engenharia de Software espacial intitulada "*Space Engineering - Software*" (Engenharia Espacial - Software) tem seu foco no software desenvolvido como parte de um sistema espacial, como um segmento espacial, lançador ou segmento de solo. Cobre todos os aspectos da Engenharia de Software espacial, ou seja, todo o ciclo de desenvolvimento de software espacial, que inclui requisitos, projeto, produção, verificação e validação, transferência, operação e manutenção.

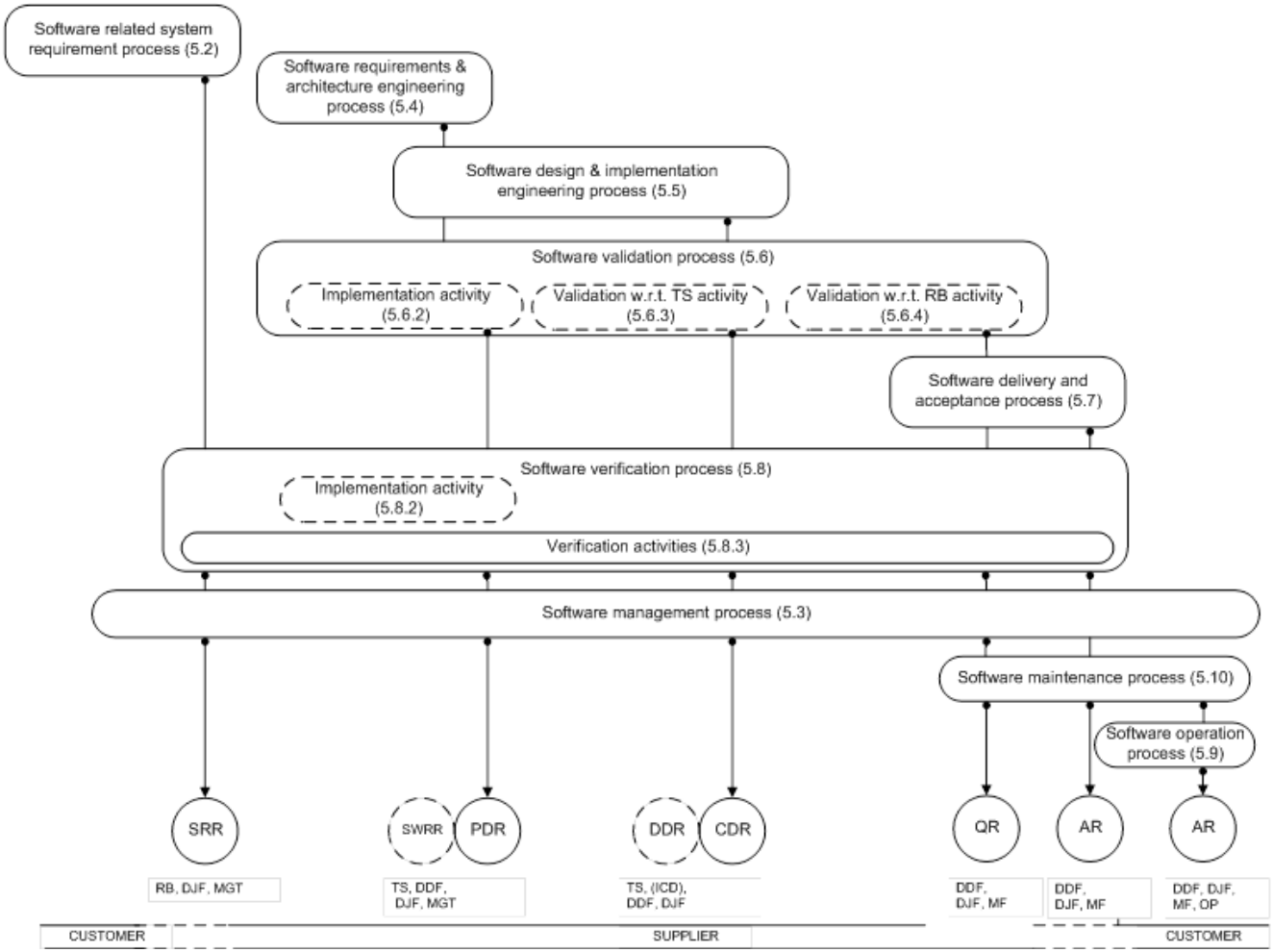
Figura 2.2 – Processos relacionados à norma ECSS-E-ST-40C




Fonte: ECSS-E-ST-40C (2009).

A E-ST-40C ressalta e faz uso do supracitado modelo cliente- fornecedor, além de refletir métodos específicos usados no desenvolvimento de sistemas espaciais.

Figura 2.3 – Ciclo de vida de software da norma ECSS-E-ST-40C.




Fonte: ECSS-E-ST-40C (2009).

As revisões são pontos relevantes em um projeto de acordo com a estrutura proposta por esta norma. Como ilustrado pela Figura 2.3, as revisões contribuem para a interação entre cliente-fornecedor e também para sincronizar os processos de Engenharia de Software.

De forma complementar, a norma E-ST-40C possui relação direta com as normas Q-ST-80C (*Software Product Assurance) e M-ST-40C (Configuration and Information Management);* e indireta com as normas Q-ST-30C (*Space Product Assurance: Dependability*) e Q-ST-40C (*Space Product Assurance: Safety*). A Figura 2.2 apresenta o relacionamento entre as normas e a estrutura de como os processos estão dispostos.

Nesse contexto, tais normas são aplicáveis somente ao processo de desenvolvimento do software, embora sejam planejadas ao nível de sistema. A Figura 2.3 também apresenta os principais processos de Engenharia de Software, mostrando em que momento as revisões ocorrem no projeto.

Conforme citado acima, o software, como parte ou componente de um sistema, deve passar por uma avaliação para definir sua categoria de criticalidade, com base na probabilidade de ocorrência e na severidade da consequência de uma falha do sistema.

Tabela 2.6 – Requisitos por categoria de criticalidade de software.

| Categoria da Criticalidade | Quantidade de Objetivos |
|---|---|
| **A** | 233 |
| **B** | 231 |
| **C** | 218 |
| **D** | 202 |

Fonte: Adaptada de ECSS-E-ST-40C (2009).

A Tabela 2.6 apresenta a quantidade de objetivos que devem ser cumpridos por categoria de criticalidade do software. Entretanto, observamos que a classificação das categorias indicadas por letras (A, B, C e D), difere da

classificação da severidade apresentada na Tabela 2.1, feita por números (1, 2, 3 e 4). A norma E-ST-40C utiliza a classificação por categorias, oriunda da norma auxiliar Q-ST-80C, a qual apresenta as categorias e a definição a partir da severidade da falha. Isso pode ser visto na Tabela 2.7 apresentada na seção 2.6.6, que descreve a norma Q-ST-80C.

Conforme observado por Reginato (2012), a norma E-ST-40C não menciona ou apresenta em seu conteúdo os níveis de independência requerida para os objetivos apresentados na Tabela 2.6. O mesmo acontece para o controle de configuração. Entretanto, os níveis de independência requerida para as atividades de verificação são definidos através norma Q-ST-80C.

### 2.6.3. ECSS-M-ST-40C

Esta norma intitulada "*Configuration and Information Management*" (Gerenciamento de Configuração e Informação), define o conjunto de requisitos necessários para o gerenciamento de configuração, informação e documentação em programas espaciais.

A norma descreve o processo de gerenciamento de configuração de forma detalhada, apresentando os requisitos para gerir informação, documentação e a configuração dentro de um programa espacial, sendo aplicada tanto ao fornecedor quanto ao cliente.

A M-ST-40C está estruturada em dois blocos:

I. Detalhamento do processo de gerenciamento de configuração e informação;

II. Objetivos a serem cumpridos.

### 2.6.4. ECSS-Q-ST-30C

Com o título de "*Space Product Assurance: Dependability*" (Confiabilidade), a norma ECSS-Q-ST-30C aborda a garantia de confiabilidade e suas exigências

para sistemas espaciais. Segundo a norma, a garantia de confiabilidade é um processo contínuo e iterativo ao longo de todo o ciclo de vida do projeto.

A política de confiabilidade para projetos espaciais é aplicada através da implementação de um programa de garantia de confiabilidade, resultando na identificação de todos os riscos técnicos com relação às necessidades funcionais que pode levar a não conformidades de acordo com as exigências de confiabilidade.

Impactos desta política no desenvolvimento de software são: a identificação de áreas críticas do projeto e a avaliação da gravidade das consequências de falhas que devem ser interpretadas pelo nível em que a análise é feita, resultando na classificação ilustrada na Tabela 2.1 (ver seção 2.2).

### 2.6.5. ECSS-Q-ST-40C

Com o título de "*Space Product Assurance: Safety*" (Segurança contra Acidentes), a norma Q-ST-40C aborda a segurança e os requisitos técnicos de segurança com o objetivo de proteger os envolvidos com o voo, suporte em terra, veículo de lançamento, o público em geral, propriedades pública e privada, sistema espaço e o ambiente dos perigos associado com sistemas espaciais.

Segundo a norma, o objetivo é garantir que todos os riscos de segurança associados com o projeto, desenvolvimento, produção e operações de produtos espaciais estejam devidamente identificados, avaliados, minimizados e controlados.

Conforme já mencionado, esta norma apresenta os objetivos para a avaliação de segurança requerida para a classificação do nível de severidade do software.

### 2.6.6. ECSS-Q-ST-80C

Com o título de "*Software Product Assurance*" (Garantia do Produto de Software), a norma Q-ST-80C define um conjunto de requisitos para garantia de produto a ser utilizada no desenvolvimento e manutenção de software para sistemas espaciais. É aplicável também a produtos não entregáveis que afetam a qualidade do produto ou serviço pertencente a um sistema espacial.

Como objetivo, esta norma busca prover confiança ao cliente e ao fornecedor de que o desenvolvimento, compra ou reuso de um software irá satisfazer os requisitos ao longo de toda vida do sistema.

A Q-ST-80C está estruturada em três blocos:

I. Implementação do programa de garantia de produto de software;

II. Garantia do processo de software;

III. Garantia de qualidade de produto de software.

Tabela 2.7 – Categoria de criticalidade de software.

| Categoria | Definição |
|---|---|
| A | Software que se não for executado, ou se não for corretamente executado, ou cujo comportamento anômalo pode causar ou contribuir para uma falha do sistema levando a **Catastróficas Consequências.** |
| B | Software que se não for executado, ou se não for corretamente executado, ou cujo comportamento anômalo pode causar ou contribuir para uma falha do sistema levando a **Críticas Consequências.** |
| C | Software que se não for executado, ou se não for corretamente executado, ou cujo comportamento anômalo pode causar ou contribuir para uma falha do sistema levando a **Grandes Consequências.** |
| D | Software que se não for executado, ou se não for corretamente executado, ou cujo comportamento anômalo pode causar ou contribuir para uma falha do sistema levando a **Pequenas ou Desprezíveis Consequências.** |

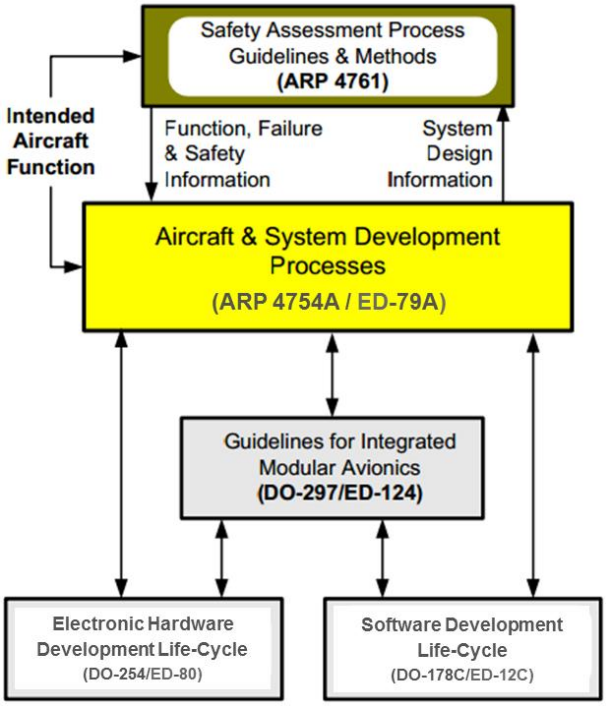
Fonte: ECSS-Q-ST-80C (2009).

Esta norma também define as categorias de criticalidade de software, conforme apresentado na Tabela 2.7, e de acordo com os níveis de severidades das falhas que, por sua vez, são definidas na norma Q-ST-30C conforme mencionado acima e apresentado na Tabela 2.1.

## 2.7. Normas da Indústria Aeronáutica

A indústria aeronáutica representa hoje uma referência importante no desenvolvimento de software crítico. Pelos numerosos desenvolvimentos de sistemas, hardwares e softwares, as normas aeronáuticas têm maior maturidade frente às normas das demais áreas.

Figura 2.4 – Diretrizes para desenvolvimento de sistemas, hardwares e softwares.




Fonte: Trivelato e Souza (2012) adaptado de SAE-ARP4754A (2012)

Conforme proposto por Trivelato & Souza (2012), a adaptação da SAE-ARP4754 apresentada na Tabela 2.4 ilustra o relacionamento entre o arcabouço de documentos de desenvolvimento, os quais provêem diretrizes para avaliação de segurança, processos do ciclo de vida de software, hardware eletrônico, desenvolvimento de sistemas e aviônica modular integrada.

### 2.7.1. SAE ARP4754A

Com o título de "*Guidelines for Development of Civil Aircraft and Systems*" (Diretrizes para Desenvolvimento de Aeronaves Civis e Sistemas), a norma aeronáutica ARP4754A (2012) aborda o ciclo de desenvolvimento de aeronaves e sistemas que implementam funções aeronáuticas.

Com ênfase em aspectos de segurança, a norma provê objetivos-chave para o desenvolvimento de aeronaves e sistemas e sugere como implementar processos para alcançar a certificação.

Segundo Reginato (2012), esta norma delineia o modelo de desenvolvimento de sistemas através de múltiplos processos de desenvolvimento de sistemas, e ainda introduz o conceito de garantia de desenvolvimento para os sistemas com uma atividade de suporte ao desenvolvimento de sistemas. Desse modo, a garantia de desenvolvimento é um processo de ações sistemáticas e especificamente planejadas.

Conforme menciona a norma, devido à natureza altamente complexa e integrada dos modernos sistemas aeronáuticos, autoridades reguladoras têm se preocupado com a possibilidade de erros culminarem e contribuírem para condições de falha na aeronave.

Devido a isto, um processo para avaliar a segurança dos sistemas é realizado como requisito pela norma SAE ARP4761. Através de práticas recomendadas como FHA (*Functional Hazard Assessment*), FTA (*Faul Tree Analysis*), FMEA (*Failure Mode and Effects Analysis*), a avaliação busca mitigar eventos indesejados e a propagação de falhas. E, uma vez estabelecida a classificação destas condições de falhas, também são estabelecidos os requisitos de segurança a serem cumpridos.

Ao longo do documento, a norma menciona a ligação com o desenvolvimento de hardware e software, discutindo o fluxo de informação e a importância dos ciclos iterativos entre eles.

### 2.7.2. SAE ARP4761

A norma aeronáutica ARP4761 publicada em 1996 e intitulado "*Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*" (Diretrizes e Métodos para Conduzir o Processo de Avaliação de Segurança contra Acidentes em Sistemas e Equipamentos Civis Embarcados), define recomendações técnicas para avaliar a segurança de sistemas embarcados.

Juntamente com a norma ARP4754A, a ARP4761 é utilizada para demonstrar o cumprimento de regulações junto a órgãos como FAA, ANAC e EASA.

De acordo com a norma, técnicas e métodos são utilizados para realizar o processo de avaliação de segurança de forma integrada com o ciclo de desenvolvimento do sistema.

O *Functional Hazard Assessment* (FHA) aborda a identificação de perigos e a análise preliminar de risco através de análise da estrutura funcional, na qual identifica e classifica as condições de falhas e seus respectivos efeitos, dando origem a ações de mitigação como alterações na estrutura funcional.

Outro método utilizado pela norma é o *Faul Tree Analysis* (FTA), no qual um diagrama lógico é construído mostrando o relacionamento dos eventos, onde uma análise busca por eventos que levem a um evento principal de falência.

Além dos métodos supracitados, a norma também apresenta e faz uso de outros métodos como *Preliminary System Safety Assessment* (PSSA), *System Safety Assessment* (SSA), *Failure Mode and Effects Analysis* (FMEA), *Failure Modes and Effects Summary* (FMES), *Dependence Diagram* (DD), *Markov Analysis* (MA).

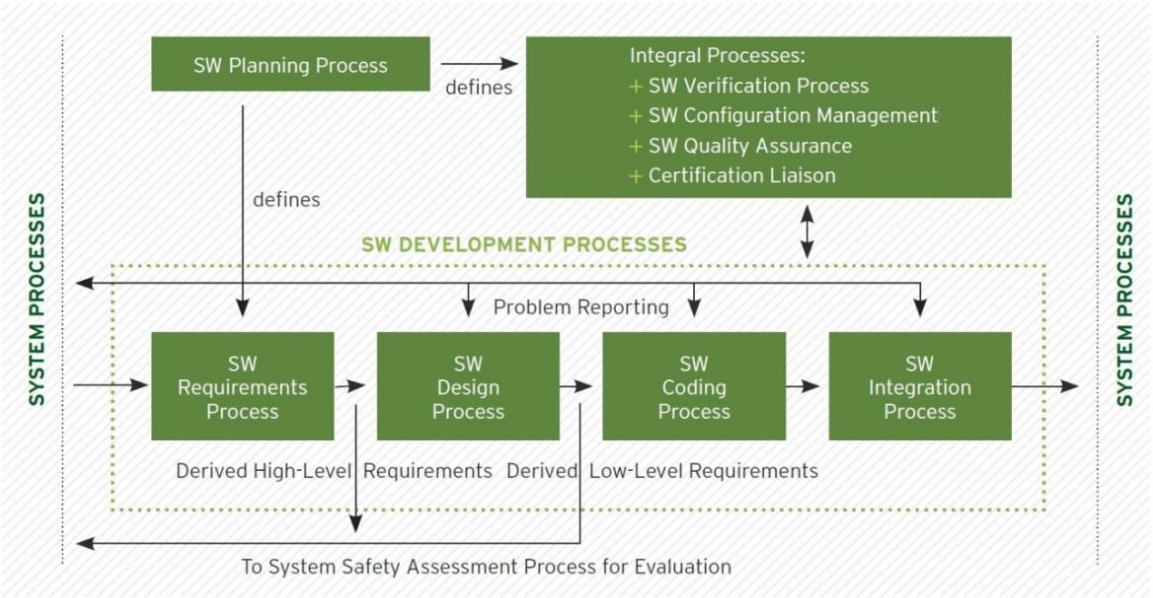
Segundo Reginato (2012), o conceito de segurança contra acidentes (*safety*) é extremamente importante para a indústria aeronáutica, considerando o risco de vida de pessoas associado a um projeto. Logo, as análises e processos

propostos pela ARP-4761 também podem ser utilizados para melhoria da confiabilidade dos sistemas espaciais.

### 2.7.3. RTCA-DO-178C

A norma RTCA-DO-178C, intitulada "*Software Considerations in Airborne Systems and Equipment Certification*" (Considerações de Software na Certificação de Equipamentos e Sistemas Embarcados) estabelecida em 2012 como sucessora da RTCA-DO-178B (1992) que já era altamente aceita pela indústria aeronáutica, fornecedores e autoridades certificadoras, como um maduro guia para obter certificação.

Figura 2.5 – Processos do ciclo de vida de um software.




Fonte: SQS (2012).

Esta norma pertence a um conjunto de normas similares da indústria aeronáutica (ver Figura 2.4 na seção 2.7) que focam nos processos de desenvolvimento e qualidade. Como principal conteúdo, a norma DO-178C apresenta a definição dos processos de desenvolvimento do ciclo de vida de software, além de processos relacionados, ilustrados na Figura 2.5. Entre as atividades relevantes podemos citar: Planejamento, Requisitos, Projeto, Codificação, Integração, Verificação, Gerenciamento de Configuração, Garantia da Qualidade, Coordenação de Certificação.

Todos os sistemas ou funções aeronáuticas possuem um **Nível de Garantia de Desenvolvimento da Função** (*FDAL – Function Development Assurance Level*) definido conforme estabelecido na ARP4754A, definido por um processo de Safety Assessment guiado pela norma ARP-4761. Também, conforme a ARP4754A, é definido o Nível de Garantia de Desenvolvimento do Item (*IDAL – Item Development Assurance Level*) a partir da decomposição em níveis. Com as categorias de condições de falha definidas e alocação destes níveis inferiores de função a hardware e software temos, consequentemente, os níveis de confiabilidade do componente de hardware e IDAL das funções de software baseado nas categorias de condições de falha.

Como apresentado na Tabela 2.8, a norma DO-178C utiliza as categorias de condições de falhas para definir os objetivos de processo a serem considerados; também apresenta a quantidade de objetivos relacionados à qualidade que precisam ser realizados com independência, significando que ao menos uma pessoa não ligada à equipe verifique estas atividades.

Tabela 2.8 – Categorização das condições de falha.

| Nível | Condição de Falha | Objetivos | Objetivos com Independência |
|---|---|---|---|
| **A** | Catastrófico | 71 | 30 |
| **B** | Perigoso | 69 | 18 |
| **C** | Maior | 62 | 5 |
| **D** | Menor | 26 | 2 |
| **E** | Sem Efeito | 0 | 0 |

Fonte: RTCA-DO-178C (2012).

A atualização da norma não se limitou somente em ser mais objetiva e clarificar questões, também trouxe quatro documentos suplementares que abordam as tecnologias mais avançadas no que tange ao desenvolvimento de software. Cada tecnologia tratada nos suplementos tem por finalidade expandir os objetivos contidos na DO-178C para aquela tecnologia, descrevendo como

cada objetivo da DO-178C será modificado ou substituído. De forma breve, se descreve a seguir os quatro suplementos:

- DO-330 - Qualificação de Ferramenta (*Tool Qualification*)

Segundo a norma DO-330, a qualificação de ferramentas é o processo usado para certificar uma ferramenta de software, necessário sempre que esta ferramenta elimine, automatize ou substitua algum processo requerido pela DO-178C. A certificação tem o propósito de garantir que a ferramenta proveja um nível de confiança no mínimo equivalente ao processo em questão.

As ferramentas são classificadas conforme dois tipos: ferramenta de desenvolvimento, que adiciona erros ao produto; ou ferramenta de verificação, que não adiciona erros mas falha em detectá-los. Uma avaliação determina o potencial impacto do uso da ferramenta no processo de desenvolvimento de software, determinando o **Nível de Qualificação da Ferramenta (TQL – *Tool Qualification Level*)**. Sendo cinco os níveis de qualificação de ferramenta indo de TQL-1 a TQL-5, sendo o primeiro mais rígido e o último o menos exigente.

- DO-331 – Desenvolvimento e Verificação Baseado em Modelos (*Model-Based Development and Verification*)

A norma guia quanto à implementação de modelos enquanto, ao mesmo tempo, assegura que o software gerado usando o modelo realiza somente as funções desejadas. Como os outros suplementos, a norma utiliza a estrutura da norma DO-178C, adicionando, modificando ou substituindo os objetivos.

Segundo a norma DO-331, um modelo é definido como uma representação abstrata de um dado conjunto de aspectos de um sistema, sendo usado para análises, verificação, simulação, geração de código ou qualquer combinação destas. A norma também define dois tipos de modelos: **modelo de especificação (*specification models*) e modelo de projeto (*design***

***models*).** O modelo de especificação representa os requisitos de software de alto nível usado para expressar o modelo funcional, desempenho, interface e características de segurança de um componente de software. O modelo de projeto os requisitos de baixo nível e a especificação da arquitetura de software para representar estruturas internas, fluxo de dados e fluxo de controle.

- DO-332 – Tecnologia de Orientação à Objetos e Técnicas Relacionadas (*Object-Oriented Technology and Related Techniques*)

O suplemento DO-332 provê um guia no uso de linguagens de programação que usem conceitos como herança, polimorfismo, sobrecarga, conversão de tipos, gerenciamento de exceções e outros conceitos. A característica que distingue a tecnologia de orientação a objetos é o uso de classes para definir objetos, e a capacidade de criar novas classes a partir de subclasses.

Conforme os demais suplementos, a norma mantém uma relação com a norma DO-178C, inclusive adicionando atividades a seus processos de desenvolvimento. A fim de garantir os objetivos de segurança contra acidentes, a norma possui um criterioso processo de verificação, isto devido às características e potenciais complicações encontradas na tecnologia de orientação a objetos.

- DO-333 – Métodos Formais (*Formal Methods*)

A norma DO-333 define métodos formais como uma notação descritiva e métodos analíticos usados para construir, desenvolver e raciocinar sobre modelos lógicos de comportamento de um sistema. O suplemento tem o objetivo de prover a adição e a modificação dos objetivos, atividades e dados do ciclo de vida de software da DO-178C para quando métodos formais forem usados no ciclo de vida de software.

Usualmente, métodos formais somente são empregados no desenvolvimento de softwares para missões críticas, negócios e segurança,

nos quais os custos de falha são altos. Métodos formais devem encontrar ou até evitar falhas que não são identificadas pelos testes, aumentando a confiança de que comportamentos anormais que possam acontecer e se mostrando complementar aos testes de software. Mas, dado que eles não podem estabelecer evidências de verificação para testes no ambiente computacional, os testes de integração de hardware/software ainda devem ser realizados para garantir que o código funcione adequadamente no ambiente computacional.

# 3 COMPARAÇÃO ENTRE NORMAS DE DESENVOLVIMENTO DE SOFTWARE

A necessidade das indústrias em entregar produtos complexos e/ou altamente integrados visando o cumprimento dos níveis de qualidade e, sobretudo, de segurança levou ao amadurecimento das normas de desenvolvimento e operação como um todo.

Hoje, sabe-se que as indústrias nuclear, aeronáutica, espacial, automotiva e médico/farmacêutica praticam abordagens de desenvolvimento semelhantes, dado o destaque das questões de segurança impostas por órgãos reguladores.

Salvo as especificidades de cada indústria, neste contexto espacial e aeronáutica, busca-se neste trabalho alinhar conceitos e abordagens de ambas as normas de desenvolvimento de software.

Nas próximas seções iremos comparar os processos, atividades e as abordagens das normas em questão.

## 3.1. Estrutura de Processos - ECSS-E-ST-40C e RTCA-DO-178C

Como objetivos intermediários deste trabalho, a Tabela 3.1 apresenta uma comparação em alto nível dos processos de desenvolvimento. Nela podemos nos apoiar para realizar uma série de importantes observações acerca dos processos contidos nas normas, principalmente no que tange à organização, disposição dos processos, abordagens e, principalmente, quanto às particularidades encontradas em cada uma das normas aqui estudadas.

Em um primeiro momento, conforme a disposição dos processos na Tabela 3.1, podemos identificar que há uma semelhança entre as estruturas de processos das normas. Entretanto alguns pontos chamam a atenção, como por exemplo, a norma E-ST-40C apresenta um processo exclusivo para a alocação dos requisitos de sistema ao software, embora a norma DO-178C não possua um processo equivalente que trate exclusivamente da alocação destes requisitos. Isto é um ponto que denota a natureza das normas aqui estudadas e

devido a esta importância nos aprofundaremos neste ponto na seção 3.3. Relação entre Sistema e Software.

Tabela 3.1 – Processos segundo as normas E-ST-40C e DO-178C.

| E-ST-40C | DO-178C |
|---|---|
| Software Management (ECSS-M-ST-10) | Software Planning |
| Software Related System Requirements | Não há processo equivalente. |
| Software Requirements And Architecture Engineering | Software Requirements |
| Software Design And Implementation Engineering | Software Design |
| | Software Coding |
| | Integration |
| Software Verification | Software Verification |
| Software Validation | |
| Software Delivery And Acceptance | Certification Liaison |
| Software Product Assurance (ECSS-Q-ST-80) | Software Quality Assurance |
| Configuration and Information Management (ECSS-M-ST-40) | Software Configuration Management |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Particularmente, a norma E-ST-40C possui alguns processos que são conduzidos por outras normas do conjunto ECSS. Segundo essa norma, este procedimento permite tratar questões mais especificamente, dada a relação com outras normas que possuem um maior aprofundamento técnico. De acordo com a Tabela 3.1, isto é visto nos processos denominados Gerenciamento de Software (*Software Management*), Garantia do Produto de Software (*Software Product Assurance*) e Gerenciamento de Configuração e Informação (*Configuration and Information Management*).

Respectivamente, o primeiro é coberto pela norma M-ST-10C do conjunto ECSS, pertencente a um subgrupo de normas de gerenciamento identificado pela letra “M” de *Management*. Este grupo específico trata de itens como cronograma, custo, risco e, sobretudo, o planejamento das atividades de software. Como seu paralelo na norma DO-178C, o processo denominado Planejamento de software (*Software Planning Process*), define o planejamento do ciclo de vida de software abordando as atividades e planos que orientam os processos de desenvolvimento e os processos integrais.

O segundo é conduzido pela norma supracitada Q-ST-80C denominada Garantia do Produto de Software e equivalente ao processo identificado como Garantia da Qualidade de Software (*Software Quality Assurance*) na norma DO-178C.

E o último é conduzido pela também supracitada norma M-ST-40C denominada Gerenciamento de Configuração e Informação e com equivalência na norma DO-178C com o processo Gerenciamento de Configuração de Software (*Software Configuration Management*).

Outra importante observação acerca dos processos denominados Entrega e Aceitação do Software (*Software Delivery And Acceptance*) e Acordo de Certificação (*Certification Liaison)*, respectivamente pertencentes às normas E-ST-40C e DO-178C, refletem a necessidade de seus setores, uma vez que os produtos do setor espacial não necessitam de certificação enquanto no setor aeronáutico civil é uma exigência.

Segundo a norma espacial, o processo Entrega e Aceitação do Software (*Software Delivery And Acceptance*) inclui atividades como: treinamento e suporte ao cliente além de testes e instalação do software no ambiente computacional. Ao cliente cabe testar o produto e fazer um aceite formal conforme o plano de aceitação definido previamente. A presença deste processo se deve basicamente ao modelo cliente/fornecedor utilizado pela norma E-ST-40C em que, ao final do desenvolvimento do software, o fornecedor conduz a entrega do produto ao cliente.

A norma DO-178C possui o processo Acordo de Certificação (*Certification Liaison)* que de forma equivalente ao processo da norma espacial, trata das atividades relacionadas a entrega do software ao processo de certificação, a qual é realizada no sistema ou equipamento que embarca o produto final de software. O processo de certificação é conduzido por uma entidade certificadora, possuindo como principais atividades: alinhamento das atividades de certificação entre autoridade certificadora e equipe candidata à certificação, definição de planos e diretrizes para a certificação e revisões para avaliar artefatos, data e evidências das atividades desenvolvidas no desenvolvimento.

De acordo com a Tabela 3.1, também podemos observar que as normas tratam a validação do software de modo distinto. A norma do setor espacial possui um processo específico denominado Validação de Software (*Software Validation*) enquanto a norma do setor aeronáutico trata a validação do software dentro do processo denominado Verificação de Software (*Software Verification*), o qual abordamos mais adiante. Para a norma E-ST-40C o processo de validação, em um primeiro momento, consiste em estabelecer o processo que valida o produto de software, o qual será utilizado em dois momentos: nas atividades de validação da Especificação Técnica antecedendo a Revisão Crítica de Projeto (*Critical Design Review - CDR*) e nas atividades de validação da Baseline de Requisitos antecedendo a Revisão de Qualificação (*Qualification Review - QR*). No que tange a norma DO-178C, esta norma não apresenta um processo exclusivo para validação do produto de software, entretanto podemos identificar atividades de validação inseridas no processo de verificação relacionados aos requisitos de software, uma vez que as validações dos requisitos de sistema alocados ao software são de responsabilidade dos processos de sistemas, como destaca a norma.

Em suma, com esta superficial comparação identificamos, com algumas poucas exceções, que os processos de ambas as normas são equivalentes. Isto evidência o alinhamento dos setores quanto a um padrão de processos, embora, distintos dados as suas inerentes características. Isto também é válido do ponto de vista que este trabalho propõe um processo que acomode estas

características em uma abordagem única. Nas próximas seções, continuaremos a traçar um paralelo entre as normas focando nos processos de desenvolvimento, sobretudo, no processo de verificação.

### 3.2. Estrutura das Normas

As normas E-ST-40C e DO-178C possuem uma evidente diferença em sua estrutura, principalmente no modo de como apresentam os objetivos a serem atingidos, atividades e artefatos de saídas.

A norma E-ST-40C tem a sua estrutura organizada em áreas de processos, as quais se desdobram em uma ou várias atividades. Estas atividades são os objetivos a serem cumpridos e estão sempre associados a uma saída esperada. A norma ainda apresenta o documento de destino desta saída esperada e o arquivo em que este documento faz parte. Também é observado que a norma não possui muitos documentos de saída, o que remete a extensos documentos e que são atualizados diversas vezes ao longo do desenvolvimento.

A estrutura apresentada pela norma dificulta consultas rápidas, embora uma tabela localizada no anexo A desta norma mostre de forma sucinta algumas informações úteis como: arquivos de entrega e seus respectivos documentos, fases em que estes documentos são atualizados e inspeções da qualidade. Esta mesma tabela não apresenta os requisitos. Ainda, podemos observar que a norma não apresenta com maiores detalhes as entradas necessárias para cada atividade. Esta ausência pode remeter a uma imprecisão no cumprimento e execução dos processos.

A norma DO-178C apresenta uma estrutura ligeiramente similar à norma E-ST-40C, porém a primordial diferença é quanto à apresentação dos requisitos ou objetivos a serem atingidos que estão organizados em tabelas contidas em anexo, o que auxilia a leitura e, sobretudo, consultas rápidas.

Estas tabelas apresentam os objetivos aplicáveis a cada nível de software (*DAL*), às atividades relacionadas e às saídas. Também apresentam dois

aspectos relevantes para cada objetivo, a independência a ser satisfeita, como apresentado na Tabela 2.8 e o rigor do controle de configuração a ser aplicado.

Finalmente, esta comparação da estrutura mostra que a norma DO-178C apresenta vantagens frente à norma E-ST-40C. A norma DO-178C apresenta uma estrutura mais enxuta e mais objetiva. O anexo em que resume os objetivos a serem cumpridos facilita a consulta e o entendimento. Isto denota a preocupação do setor com seu objetivo e, ao mesmo tempo, ser eficaz.

### 3.3. Relação entre Sistema e Software

As normas que orientam o desenvolvimento de softwares aeroespaciais E-ST-40C e DO-178C fazem parte de um arcabouço de normas para desenvolvimento de sistemas, nas quais temos, respectivamente, E-ST-10C para o âmbito espacial e ARP4754A para o aeronáutico.

O desenvolvimento de software é oriundo dos processos do ciclo de vida de sistema, a partir da alocação dos requisitos de sistema ao software. Requisitos de sistema são desenvolvidos a partir das necessidades operacionais do sistema e outras considerações como desempenho e, sobretudo, os requisitos relacionados à segurança (*safety-related*).

Com relação ao âmbito espacial, a norma E-ST-40C possui uma particularidade ao dedicar um processo para guiar o desenvolvimento de atividades de sistemas relacionadas com o software. O processo denominado Requisitos de Sistema Relacionado ao Software (*Software Related System Requirement*) produz a baseline de requisitos de sistemas que são alocados ao software. Estes requisitos são desenvolvidos com base na intenção de uso e a análise de segurança do sistema.

Dado o modelo cliente/fornecedor utilizado pela norma, o cliente é o responsável pela *baseline* de requisitos e, em particular, algumas atividades do software que compõem o sistema como, por exemplo, definir os requisitos de verificação e validação do software. Entretanto, a norma pontua que estas atividades de sistemas que são relacionadas ao software normalmente são

realizadas pelo cliente, embora possam ser delegadas ao fornecedor, mas sob orientação do cliente, justificando a presença deste processo na norma.

Para o âmbito aeronáutico, a norma DO-178C dedica, em seu conteúdo, uma pequena seção para discutir os aspectos do ciclo de vida de sistema que são necessários ao entendimento do ciclo de vida do desenvolvimento de software, sendo estes: requisitos de sistema alocados ao software, fluxo de informações entre os processos do ciclo de vida de sistemas e software, informações sobre análise de segurança, nível de software dos componentes e considerações sobre a arquitetura.

Segundo a norma, os processos de sistemas são responsáveis pela alocação dos requisitos de sistemas ao software conforme determinado pela arquitetura do sistema, os quais são refinados na fase de requisitos de software. Esta alocação dos requisitos estabelece uma fronteira e uma independência entre os processos do ciclo de vida de sistema e os processos do ciclo de vida de software, e devido a isto, ambos trocam informações de forma interativa. Esse fluxo de informações é relevante para a correção dos requisitos, arquitetura e o código fonte, e acontece ao longo do processo.

Contudo, em ambas as normas podemos identificar as fronteiras entre sistema e software e a interação entre os processos. Porém, a norma DO-178C apresenta isto de forma explícita ao estabelecer todos os dados necessários para o ciclo de desenvolvimento de software. Aqui isto é mais claro e, sobretudo, demonstra que aspectos relacionados com segurança são definidos em nível de sistemas, como por exemplo, o nível de criticalidade do componente de software.

## 3.4. Fase Conceitual

O processo de desenvolvimento de requisitos acontece de forma muito semelhante para ambas as normas, visto que nesta fase deve se produzir o conjunto de requisitos de software com base em uma análise dos requisitos de

sistemas e também dados auxiliares oriundos do processo de desenvolvimento de sistemas e processos integrais, como processos de gestão e suporte.

De acordo com a norma E-ST-40C, o subprocesso denominado Análise dos Requisitos de Software (*Software Requirements Analysis*) é parte do processo de Engenharia de Requisitos de Software e Arquitetura (*Software Requirements and Architecture Engineering Process*) e objetiva desenvolver e documentar os requisitos de software.

A norma destaca que juntamente com os requisitos de software deve-se estabelecer os requisitos de qualidade de software, pois ambos fazem parte da especificação técnica, um dos artefatos de saída do processo, como mostra a Tabela 3.2.

Segundo a norma Q-ST-80C, os requisitos de qualidade de software (*software quality requirements*) são usados para avaliar a qualidade dos requisitos de software em termos quantitativos. A partir de características como funcionalidade, disponibilidade, manutenabilidade, definem-se modelos de qualidade (*Quality Models*) os quais são usados para especificar os requisitos de qualidade software, além de outras métricas para os processos de desenvolvimento.

Dentro da análise de requisitos, a norma também orienta sobre o desenvolvimento de software modificável em vôo. Segundo a norma, o desenvolvedor deve avaliar a implicação deste na arquitetura e no processo de validação e definir requisitos funcionais e de performance, os quais devem fazer parta da especificação técnica.

Contudo, o desenvolvimento de um modelo lógico dos requisitos funcionais e de um modelo comportamental nesta fase de requisitos, formaliza e documenta os requisitos aqui desenvolvidos. Suportando a revisão denominada Revisão de Requisitos de Software (*Software Requirements Review* - SWRR), como uma antecipação da Revisão Preliminar de Projeto (*Preliminary Design Review* - PDR).

Segundo a norma DO-178C, a fase de requisitos de software (*Software Requirements*) possui os objetivos de desenvolver os requisitos de alto nível e provê-los para o Processo de Apreciação de Segurança do Sistema (*System Safety Assessment Process*), a fim de avaliar a corretude tanto dos requisitos de sistemas quanto dos requisitos de alto nível desenvolvidos.

Durante a fase de requisitos, as entradas recebidas dos processos do ciclo de vida de sistemas, como mostra a Tabela 3.2, são analisadas e utilizada no desenvolvimento dos requisitos de alto nível. Segundo a norma, os requisitos de alto nível incluem os requisitos funcionais, performance, interface e os requisitos relacionados à segurança.

Tabela 3.2 – Entradas e saídas do processo de requisitos.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Software Requirements Analysis | Software Requirements |
| **Entradas** | **Entradas** |
| • Software system specification<br>• Interface requirements document | • System Requirements<br>• Hardware Interface<br>• System Architecture |
| **Saídas** | **Saídas** |
| • Software requirements specification<br>• Software interface control document | • Software Requirements Data<br>• Trace Data |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Contudo, uma série de atividades relacionadas à gestão de requisitos está incluída no processo, como avaliar os requisitos de sistemas quanto ambiguidades e inconsistências, justificar a razão pela qual os requisitos de software existem, além de avaliar esses requisitos quanto a padrões, consistência e se são verificáveis. Estas atividades buscam a qualidade dos requisitos, dado que requisitos bem escritos são cruciais para o desenvolvimento dos casos de teste e um produto de software de qualidade. A rastreabilidade entre requisitos de sistemas e requisitos de alto nível deve ser mantida, sendo esta necessária para o processo de verificação.

No que tange a este processo, podemos observar que ambas as normas possuem um processo de desenvolvimento de requisitos muito semelhante. Entretanto, é notória a deficiência da norma espacial quanto ao detalhamento das atividades, fazendo-se necessário o uso das informações de saída como complemento. Quanto a norma aeronáutica, as informações são objetivas e trazem bom embasamento para o cumprimento dos objetivos. A Tabela 3.2 apresenta os artefatos de entrada e saída do processo de requisitos para as normas.

## 3.5. Fase de Arquitetura

O processo de arquitetura do software é apresentado de modo similar nas normas E-ST-40C e DO-178C. Entretanto, a primeira norma divide o processo em duas fases e a segunda concentra o desenvolvimento da arquitetura em apenas uma fase.

De acordo com a norma E-ST-40C, o desenvolvimento do projeto da arquitetura de software (*Software Architectural Design*) inicia-se após o desenvolvimento e a documentação dos requisitos, ainda como parte do processo denominado de Processo de Engenharia de Requisitos e Arquitetura de Software (*Software Requirements and Architecture Engineering Process*), no qual uma arquitetura preliminar do software deve ser desenvolvida.

Nesta primeira fase, são desenvolvidos itens com os diagramas sobre os aspectos de comportamento estático e dinâmico, decisões sobre interface, reuso de software e, também, definido o modelo computacional. Ao adentrar no processo denominado Processo de Engenharia de Projeto e Implementação de Software (*Software Design and Implementation Engineering Process*), temos a segunda fase da arquitetura do software chamada Projeto dos itens de software (*Design of Software Items*), onde a arquitetura desenvolvida no processo anterior é detalhada. Nesta etapa, todos os componentes de software são refinados em mais baixo nível, permitindo a transformação da arquitetura de componentes em unidades que permitem a codificação.

Precedendo a revisão formal, um volumoso processo de documentação se inicia, onde então podemos destacar o Manual do Usuário do Software (*Software User Manual*) e o Plano de Teste das Unidades de Software (*Software Unit Test Plan*). No primeiro, o software é detalhado em termos de uso e instalação definindo responsabilidades, cronogramas, procedimentos; e no segundo são definidas as abordagens e a verificação do software.

Segundo a norma DO-178C, o processo de desenvolvimento da Arquitetura de Software (*Software Design*) é realizado a partir de um refinamento dos requisitos de alto nível de software. Este processo é conduzido de forma iterativa, onde o desenvolvimento pode proceder com uma ou mais iterações. Ao final desse processo teremos a arquitetura do software e os requisitos de baixo nível, os quais podem ser utilizados no desenvolvimento do código fonte. Ainda, conforme orienta a norma, as saídas desse processo devem ser providas para o processo de sistemas e avaliação de segurança, assim como supracitado.

Tabela 3.3 – Entradas e saídas do processo de arquitetura de software

| **E-ST-40C** | **DO-178C** |
|---|---|
| Software Architectural Design / Design of Software Items | Software Design Process |
| **Entradas** | **Entradas** |
| • Software Requirements Specification<br>• Software Interface Control Document | • Software Requirements Data |
| **Saídas** | **Saídas** |
| • Software Design Document<br>• Software Interface Control Document<br>• Software Reuse File<br>• Software Unit Test Plan<br>• Software User Manual | • Software Architecture<br>• Low-Level Requirements |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Cabe aqui mencionar que a norma DO-178C, mais uma vez enfatiza a segurança do desenvolvimento ao destacar e sugerir o cumprimento de

atividades extras para o Projeto de Software Modificável pelo Usuário (*Designing for User-Modifiable Software*) e Software com Código Desativado (*Designing for Deactivated Code*), desta forma prevenindo quaisquer impactos desses na segurança operacional.

A Tabela 3.3 apresenta as entradas e saídas dos processos referentes à arquitetura de software. Haja vista que a norma espacial apresenta as saídas deste processo e, entre eles, um artefato referente ao plano de verificação, enquanto a norma aeronáutica apresenta somente os artefatos de saídas referente a arquitetura. Isto acontece porque a norma aeronáutica mantém o processo de verificação em paralelo, produzindo os respectivos artefatos, mas como saída de outro processo que não relacionamos neste momento. Contudo, apesar de as estruturas dos processos serem distintas quanto às etapas, ao final da fase de arquitetura do software temos a arquitetura dos componentes definidos.

## 3.6. Fase de Implementação

Os processos de implementação ou codificação praticados pelas normas E-ST-40C e DO-178C compartilham de poucas características, embora o desenvolvimento do código seja realizado com base nos requisitos de baixo nível e arquitetura de software.

Na norma E-ST-40C, o sub processo denominado Código e Teste (*Coding and Testing*), ainda pertencente ao processo denominado de Engenharia de Projeto de Implementação e Software (*Software Design and Implementation Engineering*) aborda o desenvolvimento e a documentação das unidades de software, procedimentos para compilação e a ligação entre elas.

Ainda no mesmo subprocesso, na sequência a norma aborda os testes das unidades de software. O fornecedor deve desenvolver e documentar os procedimentos de teste e os dados que serão utilizados nos testes de cada unidade. Os testes unitários devem satisfazer aos requisitos e produzir os

relatórios de teste. A Tabela 3.4 identifica os artefatos de saída desta fase de implementação.

Quanto aos testes de unidade de software realizados nesta fase, segundo a norma devem exercitar: repetições, mensagens de erros, variáveis globais como definidas nos documentos de projeto e testes de entradas para verificar o comportamento anormal do software quanto valores indesejados. Mais adiante trataremos com detalhes sobre os testes de unidade de software.

Segundo a norma DO-178C, a implementação dos requisitos de baixo nível é realizada conforme a arquitetura de software e seguindo um padrão de codificação de software. O desenvolvimento dos procedimentos de compilação, ligação entre unidades e o carregamento no hardware é abordado no processo de integração conforme mencionado na norma. Contudo, a norma também aborda que o uso da geração automática de código deve estar de acordo com as restrições definidas no processo de gestão.

Tabela 3.4 – Entradas e saídas do processo de implementação

| **E-ST-40C** | **DO-178C** |
|---|---|
| Coding and Testing | Software Coding |
| **Entradas** | **Entradas** |
| • Software Design Document<br>• Software Interface Control Document<br>• Software Reuse File | • Software Architecture<br>• Low-Level Requirements |
| **Saídas** | **Saídas** |
| • Software Component Design Documents and Code<br>• Source Code<br>• Software Configuration File<br>• Software Unit Test Plan<br>• Software Unit Test Report | • Source Code<br>• Trace Data |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

A norma DO-178C novamente ressalta que qualquer entrada inadequada ou incorreta identificada no processo de codificação deve ser provida para os processos anteriores, como: Processo de Requisitos de Software, Processo de Projeto de Software e Processo de Planejamento de Software. Esta realimentação tem por finalidade clarificar e corrigir erros identificados, nitidamente exemplificando a prática do conceito de processo de desenvolvimento iterativo.

Embora, o processo de implementação de forma geral seja similar e brevemente mencionado em ambas as normas, podemos identificar alguns pontos divergentes entre eles. A norma espacial realiza algumas atividades de integração após a implementação das unidades enquanto a norma aeronáutica as mantém sob o processo de integração. Isto também acontece com os testes unitários praticados pela norma espacial dentro do processo de implementação e mantido em paralelo pela norma aeronáutica no processo de verificação, ambos visto adiante. Também é visto que ambas as normas mencionam o uso de um padrão de codificação, sendo isto diretamente colocado na norma aeronáutica, enquanto que para a norma espacial isto é mencionado através da norma Q-ST-80C.

## 3.7. Fase de Integração

O processo de integração é abordado nas normas DO-178C e E-ST-40C seguindo propostas distintas, expressando a abordagem seguida por cada uma.

A norma E-ST-40C apresenta o subprocesso de integração (*Integration*) como a última parte do processo Engenharia de Projeto de Software e Implementação (*Software Design and Implementation Engineering*). De acordo com o processo, em um primeiro momento é detalhado o plano de testes de integração de software para definir a integração das unidades e dos componentes, a fim de torná-los um item de software. Este plano deve conter informações sobre o projeto de testes, a especificação dos casos de teste, os procedimentos de teste e dados de teste.

Na sequência, o processo orienta para a integração das unidades de software e componentes de software. Os testes são realizados após isto, os quais devem atender aos requisitos do item de software. Como apontado pela Tabela 3.5, ao final da integração o plano e o relatório de integração são emitidos. Haja vista que o processo não aborda a integração hardware/software, o qual é tratado no processo denominado Entrega e Aceitação do Software (*Software Delivery And Acceptance*).

Para a norma DO-178C, o objetivo do processo intitulado Integração (*Integration*) é integrar o código fonte com o hardware (*target*), para produzir o sistema integrado ou o equipamento.

Embora, segundo a norma, o objetivo seja integrar hardware/software, como objetivo intermediário do processo está a integração dos componentes para termos um item de software. A partir do qual, juntamente com o compilador, *linking* e dados de carregamento, são gerados o Código Objeto (*Object Code*) Código Objeto Executável (*Executable Object Code*). Estas são as saídas do processo, como mostrado pela Tabela 3.5.

Tabela 3.5 – Entradas e saídas do processo de integração.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Integration | Integration |
| **Entradas** | **Entradas** |
| • Software Component Design Documents and Code<br>• Source Code | • Software Architecture (Design Process)<br>• Source Code |
| **Saídas** | **Saídas** |
| • Software Integration Test Plan<br>• Software Integration Test Report | • The Object Code<br>• Executable Object Code<br>• Parameter Data Item<br>• Compiling Data<br>• Linking Data<br>• Loading Data |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Em resumo, os processos de integração conforme apresentados pelas normas e aqui expostos, possuem objetivos distintos. A norma espacial trata em seu processo basicamente a integração de software, entre unidades e componentes. Não está nesse escopo a integração de hardware / software, a qual trata disto mais adiante. Já a norma aeronáutica, aborda em seu processo a integração de componentes e a integração de hardware / software. Esta diferença entre os processos de integração reflete a abordagem utilizada para cada uma das normas. Mais adiante nos aprofundaremos nestas abordagens.

## 3.8. Processo de Verificação

Aqui iremos detalhar o processo e as atividades de verificação que são realizadas ao longo do desenvolvimento do software.

Segundo Peña e Souza (2013), o processo de verificação e validação e, por extensão, certificação, vem se tornando uma fase extremamente importante do ciclo de vida de um projeto, dada o surgimento do software crítico (*safety-critical software*) cuja falha pode resultar em perdas humanas.

Em ambas as normas o processo de verificação cumpre um efetivo papel na qualidade e confiabilidade dos produtos de software, detectando e reportando erros que podem ser introduzidos durante o processo de desenvolvimento.

Presente em todas as fases do ciclo de vida e acontecendo em paralelo ao processo de desenvolvimento, o processo de verificação inicia-se na fase conceitual e finda na entrega do produto.

Sobre a norma E-ST-40C, o processo de verificação possui dois objetivos. O primeiro é a implementação do processo de verificação, no qual o fornecedor define o processo de verificação e a organização responsável por conduzir o processo. O segundo que são as atividades de verificação, as quais têm início na Revisão de Requisitos (SRR) sendo executada pelo cliente. Para o fornecedor o processo inicia-se com o desenvolvimento do Plano de Verificação, o que acontece antes da revisão de projeto preliminar (PDR). O

resultado do processo de verificação é acompanhado a cada revisão através de relatórios.

Esta mesma norma, também menciona que o cumprimento do processo de verificação pode ser executado com vários níveis de independência, podendo ser realizado por diferentes pessoas na organização ou por organizações diferentes, o que a norma denomina como Verificação e Validação Independente de Software (*Independent Software Verification and Validation*).

Logo, dois pontos merecem destaque sobre o processo de verificação apresentado na norma E-ST-40C: o primeiro é que as atividades de verificação presentes no processo são realizadas conforme a análise de certos itens; e o segundo, é que devido ao processo denominado Requisitos de Sistema Relacionados ao Software (*Software Related System Requirements*), a norma apresenta um processo de verificação voltado aos requisitos e outro voltado à verificação da especificação técnica, ambos descritos adiante.

Segundo a norma DO-178C, verificação não é simplesmente teste. Em geral, testes não mostram ausência de erros, dado que a norma utiliza o termo “verificar” ao longo do processo de verificação. A verificação consiste de uma combinação de revisões, análises e testes. Este último se desdobra em: desenvolvimento de casos de testes e procedimentos de testes e a execução destes. Segundo estes três tipos de verificação, revisões e análises proporcionam uma avaliação da completude dos requisitos, arquitetura de software e código fonte. Enquanto o desenvolvimento dos casos de teste e procedimentos de teste avaliam a consistência e a completude dos requisitos, a conformidade com os requisitos é provida com a execução dos procedimentos de teste.

De acordo com a norma, revisões e análises são aplicadas às saídas do ciclo de desenvolvimento de software. Uma **revisão** consiste da inspeção das saídas de um processo guiada por um *checklist*, enquanto uma **análise** deve examinar em detalhes a funcionalidade, desempenho e a implicações de segurança de um componente de software, bem como o relacionamento deste

componente dentro do sistema ou equipamento. Ainda, podem haver alguns casos em que os objetivos de verificação não sejam totalmente satisfeitos com a revisão e análises sozinhas; neste caso, estes objetivos devem ser satisfeitos com testes de software.

O processo de verificação da norma DO-178C possui como saídas os seguintes artefatos: *Software Verification Cases and Procedures*, *Software Verification Results* e A*ssociated Trace Data*. Podemos ver mais adiante que estes artefatos são as saídas associadas a cada etapa do processo de verificação, os quais são atualizados conforme a estas etapas.

Um aspecto importante também apontado pela norma é o cumprimento das atividades do processo de verificação com independência, ou seja, a verificação de um determinado item deve ser realizada por uma pessoa que não está diretamente envolvida com o desenvolvimento deste item. Por fim, a norma também pontua quanto ao uso de ferramentas nas atividades de verificação, uma vez que auxilia alcançar um melhor desempenho nas atividades.

### 3.8.1. Verificação na Fase Conceitual

A norma E-ST-40C decompõe o processo de verificação de requisitos em dois momentos. Em um primeiro momento, a verificação atua nos requisitos de sistemas alocados ao software e depois nos requisitos de software derivados. Eles são desempenhados respectivamente pela equipe de sistemas e pela equipe de software ou cliente e fornecedor.

Como supracitado, a parte inicial denominada Verificação da *Baseline* de Requisitos (*Verification of Requirements Baseline*) apresenta as atividades que compõem a verificação, entre as quais podemos destacar as seguintes: descrição do ambiente em que o software opera, especificação de todos os sistemas externos que interagem com o produto de software, especificar modos, submodos e a transição entre eles, definição do cenário operacional e inclusão de requisitos consistentes e verificáveis. O artefato Relatório de

Verificação da *Baseline* de Requisitos (*Requirements Baseline Verification Report)* é a saída esperada das atividades de verificação para esta etapa como apresenta a Tabela 3.6.

A segunda parte do processo de verificação denominado verificação da especificação técnica (*Verification of the Technical Specification*) remete aos requisitos derivados dos requisitos de sistema. Dentre as atividades destacam-se rastreabilidade entre requisitos de sistemas e software, verificabilidade dos requisitos de software e identificação das restrições de implementação e ambiente de hardware. Os artefatos Matrizes de Rastreabilidade de Requisitos (*Requirements Traceability Matrices)* e o Relatório de Verificação de Requisitos (*Requirements Verification Report)* são as saídas esperadas das atividades de verificação para esta etapa.

Tabela 3.6 – Entradas e saídas do processo de verificação na fase conceitual.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Verification of Requirements Baseline / Verification of the Technical Specification | Reviews and Analyses of High-Level Requirements / Reviews and Analyses of Low-Level Requirements |
| **Entradas** | **Entradas** |
| • Software requirements specification<br>• Software interface control document | • Software Requirements Data<br>• Trace Data |
| **Saídas** | **Saídas** |
| • Requirements Baseline Verification Report<br>• Verification of the Technical Specification | • Software Verification Cases and Procedures<br>• Software Verification Results<br>• Associated Trace Data. |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Segundo a norma DO-178C, a verificação das saídas da Fase Conceitual é conduzida através de revisões e análises, as quais buscam identificar e reportar erros introduzidos durante o processo. A verificação é realizada em duas etapas denominadas Revisões e Análises dos Requisitos de Alto Nível

(*Reviews and Analyses of High-Level Requirements*) e Revisões e Análises dos Requisitos de Baixo Nível (*Reviews and Analyses of Low-Level Requirements*).

Devido às semelhanças dos itens a serem verificados, os objetivos para ambos os processos são os mesmos, a saber: cumprimento dos requisitos de sistema e requisitos de alto nível (*compliance with system requirements / High-Level*), acurácia e consistência (*accuracy and consistency*), compatibilidade com o computador escolhido (*compatibility with the target computer*), verificável (*verifiability*), em conformidade com a norma (*conformance to standard*), rastreabilidade (*traceability*) e aspectos do algoritmo (*algorithm aspects*).

A verificação para a fase conceitual é semelhante segundo a comparação das normas. Podemos observar que existe uma decomposição e as atividades acontecem em duas etapas, atendendo aos requisitos de alto nível; e, na sequência, aos requisitos de baixo nível. Analises e revisões são usadas na verificação. Quanto aos artefatos de saída, a norma aeronáutica utiliza os mesmos artefatos durante todo o processo, atualizando-os conforme cada etapa do processo de verificação, como supracitado.

### 3.8.2. Verificação na Fase de Arquitetura

Conforme supracitado, a norma E-ST-40C organiza os processos de arquitetura de software em duas etapas: a primeira aborda a arquitetura em alto nível, enquanto a segunda aprofunda e detalha esta arquitetura. O processo de verificação da arquitetura segue com as mesmas características, as quais serão descritas na sequência.

Nesta primeira etapa da verificação da arquitetura de software denominada Verificação do Projeto da Arquitetura do Software (*Verification of the Software Architecture Design*), a norma E-ST-40C continua a realizar a verificação através da análise de itens, dentre os quais se destacam: consistência interna entre os componentes de software, rastreabilidade entre os requisitos e os componentes de software, justificativa dos requisitos que não são rastreados, correção da arquitetura desenvolvida com relação aos requisitos e interfaces,

sincronização entre interfaces externas e os tempos internos e, por fim, a viabilidade de produzir a arquitetura detalhada. Os artefatos Matriz de Rastreabilidade de Requisitos para o Projeto de Arquitetura de Software (*Software Architectural Design to Requirement Traceability Matrices)* e Relatório de Verificação da Interface e Arquitetura do Projeto de Software (*Software Architectural Design and Interface Verification Report)* são as saídas esperadas das atividades de verificação para esta etapa, como apresentado na Tabela 3.7.

Tabela 3.7 – Entradas e saídas do processo de verificação na fase de arquitetura.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Verification of the Software Architecture Design / Verification of the Software Detailed Design | Reviews and Analyses of Software Architecture |
| **Entradas** | **Entradas** |
| • Software Design Document<br>• Software Interface Control Document | • Software Architecture |
| **Saídas** | **Saídas** |
| • Software Architectural Design to Requirement Traceability Matrices<br>• Software Architectural Design and Interface Verification Report<br>• Detailed design traceability matrices<br>• Detailed design verification report | • Software Verification Cases and Procedures<br>• Software Verification Results<br>• Associated Trace Data. |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Seguindo com a segunda etapa da verificação da arquitetura denominada Verificação do Projeto Detalhado de Software (*Verification of the Software Detailed Design*), agora em um nível mais detalhado, a norma realiza, através da análise de itens, a verificação dos artefatos produzidos. Entre os itens verificados destacam-se os seguintes: arquitetura detalhada consistente com a arquitetura de alto nível, completude da rastreabilidade entre a arquitetura e a arquitetura detalhada, justificativa das unidades de software que não são rastreadas para os componentes e viabilidade de realizar os testes de software.

Esta etapa da verificação produz as seguintes saídas esperadas: Matrizes de Rastreabilidade de Projeto Detalhado (*Detailed design traceability matrices)* e Relatório de Verificação do Projeto Detalhado (*Detailed design verification report),* como mostra a Tabela 3.7.

De forma similar a norma DO-178C utiliza revisões e análises para identificar se os artefatos produzidos na fase de arquitetura de software atende aos objetivos de verificação. Destacamos aqui alguns destes objetivos: compatibilidade com os requisitos de alto nível (*compatibility with the high-level requirements*), consistência (*consistency*), compatibilidade com o computador escolhido (*compatibility with the target computer*), verificabilidade (*verifiability*), em conformidade com a norma (*conformance to standard*), integridade no particionamento (*Partitioning Integrity*).

De acordo com as normas, podemos observar que, em ambas, a verificação da arquitetura é realizada de forma similar, ocorrendo conforme alguns critérios e através de revisões e análises.

### 3.8.3. Verificação na Fase de Implementação

Conforme notado na fase de Implementação (ver seção 3.6), que a norma associa no mesmo processo a codificação e os testes de unidade, aqui no processo de verificação isto também é observado. A verificação na fase de implementação, segundo a norma E-ST-40C, é composta pelas etapas denominadas Verificação de Código (*Verification of Code*) e Verificação dos Testes Unitários de Software (*Verification of Software Unit Testing*).

Na primeira etapa o fornecedor analisa o código fonte do software verificando itens como: consistência entre as unidades do software, rastreabilidade do código com a arquitetura e requisitos, justificativa sobre o código não rastreado com as unidades, implementação de mecanismos de proteção numérica, implementação de requisitos críticos e de segurança e inexistência de estouro de memória. Esta verificação produz como saídas esperadas os seguintes artefatos: Matrizes de Rastreabilidade de Código de Software (*Software Code*

*Traceability Matrices)* e Relatório de Verificação de Código de Software (*Software Code Verification Report*), as quais podem ser vistas na Tabela 3.8.

Ainda na primeira etapa e de forma complementar, o fornecedor verifica a cobertura de código e a robustez do software. A cobertura de código é verificada a partir dos resultados da execução dos testes unitários, testes de integração e testes de validação, enquanto a verificação da robustez deve ser realizada através de testes do tipo caixa branca ou análise estática para casos onde não seja possível detectar os erros em tempo de execução. Estes itens complementares produzem como saídas os artefatos Relatório de Verificação de Cobertura de Software (*Code coverage verification report)* e Relatório de Verificação de Robustez (*Robustness verification report)*.

Na segunda etapa da verificação na fase de implementação, o fornecedor verifica os resultados dos testes de unidade para garantir que: testes de unidades são consistentes com a arquitetura detalhada e os requisitos, testes de unidades são rastreáveis com requisitos de software, integração de software e testes são viáveis, resultados dos testes estão conforme o resultado esperado, dados dos testes são mantidos sob a gestão de configuração. Nesta etapa as saídas esperadas são: Matrizes de Rastreabilidade de Testes de Unidade de Software (*Software unit tests traceability matrices)* e Relatórios de Verificação de Testes Unitários de Software (*Software unit testing verification report*), como mostra a Tabela 3.8.

De acordo com a norma DO-178C, revisões e análises atuam para detectar e reportar os erros introduzidos durante a codificação. As principais correções realizadas pelas revisões e análises de código são relativas aos requisitos de software, arquitetura de software e a conformidade com o padrão de codificação.

As atividades de revisões e análises confirmam se o código fonte satisfaz aos seguintes objetivos: cumprimento com os requisitos de baixo nível, cumprimento com os requisitos de alto nível, verificável, rastreabilidade, precisão e consistência. Entre os objetivos destaca-se a conformidade com um

padrão de codificação, o qual guia quanto à complexidade e restrições no desenvolvimento do código fonte.

Nesta fase de verificação, a norma espacial chama a atenção pela abordagem de verificação complementar associada aos resultados dos testes de software, uma vez que as atividades de verificação dos resultados de cobertura, testes de robustez e testes unitários deveriam estar associadas aos processos de testes unitários, integração e validação, denotando deste modo que há uma forte interação entre estes processos. Quanto à norma aeronáutica, as atividades de verificação seguem as mesmas características encontradas até esta fase, analises e revisões verificando o cumprimento dos objetivos.

Tabela 3.8 – Entradas e saídas do processo de verificação na fase de implementação.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Verification of Code / Verification of Software Unit Testing | Reviews and Analyses of Source Code |
| **Entradas** | **Entradas** |
| • Software Component Design Documents and Code<br>• Source Code<br>• Software Configuration File | • Source Code<br>• Trace Data |
| **Saídas** | **Saídas** |
| • Software Code Traceability Matrices<br>• Software Code Verification Report<br>• Software unit tests traceability matrices<br>• Software unit testing verification report | • Software Verification Cases and Procedures<br>• Software Verification Results<br>• Associated Trace Data. |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

### 3.8.4. Verificação na Fase de Integração

Segundo a norma E-ST-40C, no processo denominado Verficação do Integração do Software (*Verification os Software Integration*) o fornecedor do item de software deve verificar se a integração foi realizada de acordo com a estratégia do plano de integração e atividades de integração. A verificação busca o cumprimento de itens como: rastreabilidade com a arquitetura de

software; consistência interna; objetivos dos testes de interface e a conformidade com os resultados esperados. Este processo tem como única artefato de saída esperado o *Software integration verification report – SVR,* como apresentado na Tabela 3.9.

Para a norma DO-178C, revisões e análises das saídas do processo de integração são realizadas a fim de detectar e reportar erros. A verificação neste ponto atua em um único objetivo que, segundo a norma, é garantir que as saídas do processo de integração estejam completas e corretas. E assim, incluindo atividades que examinem detalhadamente a compilação, *Linking*, carregamento de dados, e ainda, o mapa de memória. Sobre a verificação no processo de integração, ambas as normas procedem de modo similar.

Tabela 3.9 – Entradas e saídas do processo de verificação na fase de integração.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Verification os Software Integration | Reviews and Analyses of the Outputs of the Integration Process |
| **Entradas** | **Entradas** |
| • Software Component Design Documents and Code<br>• Software Integration Test Plan | • The Object Code<br>• Executable Object Code<br>• Parameter Data Item<br>• Compiling Data<br>• Linking Data<br>• Loading Data |
| **Saídas** | **Saídas** |
| • Software integration verification report | • Software Verification Cases and Procedures<br>• Software Verification Results<br>• Associated Trace Data. |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

## 3.9. Verificação por Testes

Juntamente com as revisões e análises, os testes de software são usados na verificação de software para demonstrar que o código fonte satisfaz os

requisitos e, também demonstrar, com um alto grau de confiança, que os erros que poderiam levar a uma indesejável e inaceitável condição de falha foram removidos.

As normas E-ST-40C e DO-178C apresentam diferentes abordagens para os testes de software. Em ambas é possível identificar que os testes são realizados conforme três níveis, a saber: Testes de unidade, testes de integração de software e testes de integração de hardware e software. Observamos que dado o contexto de atuação, diferentes termos e abordagens são identificados. Nesta seção, serão apresentadas as abordagens e particularidades sobre os processos de testes das normas aqui estudadas.

A norma E-ST-40C apresenta uma abordagem de testes que se inicia com os testes de unidades, avança para os testes de integração das unidades e finaliza com os testes de integração de hardware / software. As atividades de testes estão fracionadas em diferentes processos, sendo cumpridas juntamente com o cliente em alguns momentos. A norma Q-ST-80C é mencionada em vários momentos neste processo, orientando quanto a critérios que devem ser cumpridos nas atividades e ser, posteriormente, avaliados através das revisões de qualidade.

E-ST-40C: Testes de unidade: As atividades que guiam os testes unitários pertencem ao processo denominado Engenharia de Projeto e Implementação de Software (*Software Design and Implementation Engineering)*. Segundo esta norma, após o desenvolvimento das unidades de software, o fornecedor deve desenvolver e documentar os procedimentos de teste, testar cada unidade para garantir a satisfação dos requisitos de baixo nível de software e, por fim, documentar os resultados.

Os testes unitários exercem um papel de verificação do código quanto à alguns critérios os quais, caso apresentem problemas, podem impactar as atividades de integração das unidades e componentes. Logo, de acordo com esta norma, os testes de unidade devem exercitar os seguintes itens: laços de repetição e comparações, mensagens e casos de erros definidos no projeto, acesso a

variáveis globais, valores Limites (teste de estresse), Valores inválidos (dados de entrada fora do intervalo). Segundo a norma E-ST-40C, os artefatos de saída para esta etapa dos testes são:

- *Software component design document and code.*
- *Software unit test plan.*
- *Software component design document and code.*
- *Software unit test reports.*
- *Software unit test reports.*

E-ST-40C: Testes de integração: São abordados na atividade de Integração, a qual também pertence ao processo Engenharia de Projeto e Implementação de Software (*Software Design and Implementation Engineering*) e após a integração das unidades de software e componentes de software. Realizados pelo fornecedor e com base no plano de integração, os testes buscam garantir que o item de software satisfaça os requisitos e a integração ao final da atividade. Com relação a detalhes de execução dos testes, a norma não apresenta uma relevante profundidade para descreve as atividades que devem ser desenvolvidas ou, até mesmo, quais testes e técnicas devem ser aplicadas. A norma E-ST-40C aponta os seguintes artefatos como saídas para esta etapa dos testes:

- *Software integration test plan.*
- *Software integration test report.*

E-ST-40C: Testes de integração de hardware/software: Os testes são guiados pela atividade denominada Aceitação de Software (*Software Acceptance*) como parte do processo Entrega e Aceite de Software (*Software Delivery and Acceptance*). Dado o modelo cliente/fornecedor utilizado pela norma espacial, neste processo, o fornecedor prepara a entrega do software e a aplicação dos testes no ambiente computacional do cliente.

No escopo dos testes de aceitação está o planejamento das atividades, através da criação de um plano que define os testes selecionados para o ambiente computacional. Após a execução dos testes de integração entre o software e o ambiente computacional, o fornecedor deve gerar o código executável a partir das configurações dos componentes e instalá-lo no ambiente computacional do cliente. Ainda como parte da aceitação, o fornecedor auxilia o cliente a realizar a revisão de aceitação e os testes do software, os quais devem ter os resultados documentados. Por fim, os testes de aceitação devem ser rastreados à *baseline* de requisitos. Segundo a norma E-ST-40C, os seguintes documentos são as saídas das atividades de testes para esta última etapa:

- *Acceptance test plan.*
- *Acceptance test report.*
- *Software product.*
- *Joint review reports.*
- *Traceability of acceptance tests to the requirements baseline.*

A norma DO-178C concentra as atividades de Testes de Software no processo denominado Verificação de Software (*Software Verification*). Esta norma apresenta uma abordagem de testes de software com três níveis, sendo eles: testes de integração hardware/software, testes de integração de software e testes de unidades. Segundo a norma, o objetivo dos testes é a execução do software para confirmar o cumprimento dos requisitos, robustez e compatibilidade com o ambiente computacional (*target*).

Devido à sequência de testes apresentada pela abordagem, inicialmente a norma propõe um teste para verificar a compatibilidade com o ambiente computacional, o qual inclui o carregamento do software em um ambiente semelhante ao de destino, o qual pode ser emulado ou simulado. Segundo a norma, a realização dos testes no próprio ambiente computacional demonstra erros que somente são detectados no próprio ambiente.

A norma utiliza um método de testes na qual os casos de teste são criados com base nos requisitos e atendendo a dois tipos: Classe Válida e Classe Inválida. A execução dos testes com base nos casos de testes e procedimentos de testes são conforme os três níveis. Conforme supracitado, a norma centraliza todos os dados gerados durante o processo de verificação em apenas três artefatos, os quais são atualizados ao longo das atividades do processo de verificação, a saber:

- *Software Verification Cases and Procedures*
- *Software Verification Results*
- *Associated Trace Data.*

DO-178C: Testes de integração hardware/software: Os testes buscam satisfazer os requisitos de alto nível, abordando erros provenientes da operação do software no ambiente computacional e funcionalidades de alto nível. Segundo a norma, alguns erros tipicamente encontrados por este método são:

- Falha em requisitos quanto ao tempo de execução.
- Barramento de dados com problemas.
- Erros na interface de hardware e software.
- Estouro de pilha.

Os testes de integração hardware / software também são responsáveis por realizar atividades de verificação que estão no escopo de sistemas. Isto pode ser evidenciado pelas entradas recebidas dos processos de sistemas e, também, através dos erros típicos encontrados, os quais, por natureza, estão no escopo de sistemas, por exemplo, mapeamento de memória.

DO-178C: Teste de integração de software: Os testes realizados neste nível se concentram no inter-relacionamento entre os requisitos de software e a

implementação dos requisitos pela arquitetura de software, garantindo que uma correta relação entre os componentes de software satisfaz os requisitos e a arquitetura de software. Para este nível de teste, segundo a norma os erros típicos encontrados são:

- Incorreta inicialização de variáveis e constantes.
- Erros na passagem de parâmetros.
- Dados corrompidos.
- Incorreta sequência de eventos e operações.

DO-178C: Teste de baixo-nível de software: os testes realizados neste nível buscam demonstrar que cada componente e unidade de software cumprem com os requisitos de baixo nível. A norma aponta que os típicos erros encontrados neste nível são:

- Falha de um algoritmo para satisfazer um requisito de software.
- Laço de repetição Incorreto.
- Lógica de decisão incorreta.
- Falha em processar corretamente combinação de condições de entradas.
- Resposta incorreta para a entrada de dados faltante ou corrompida.
- Algoritmo com inadequada precisão, exatidão e performance.

Após a aplicação dos procedimentos de testes, duas análises avaliam a efetividade e cobertura dos testes. A primeira avalia os casos de teste com relação aos requisitos de software, verificando se os casos de testes selecionados cumpriram os critérios definidos. E a segunda avalia se os casos de teste exercitaram a estrutura do código satisfazendo os critérios de cobertura. Caso as análises identifiquem que os testes não atendam aos

critérios de cobertura, novos casos de testes baseados nos requisitos devem ser especificados, gerados e aplicados.

Tabela 3.10 – Entradas e saídas do processo de verificação por testes.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Software Design and Implementation Engineering / Software Design and Implementation Engineering / Software Delivery and Acceptance | Software Testing |
| **Entradas** | **Entradas** |
| • Software Component Design Documents and Code<br>• Software unit test plan<br>• Software Integration Test Plan<br>• Acceptance test plan<br>• Software Product | • Software Verification Cases and Procedures<br>• The Object Code<br>• Executable Object Code<br>• Parameter Data Item<br>• Compiling Data<br>• Linking Data |
| **Saídas** | **Saídas** |
| • Software component design document and code<br>• Software unit test plan<br>• Software component design document and code<br>• Software unit test reports<br>• Software unit test reports<br>• Software integration test plan<br>• Software integration test report<br>• Acceptance test plan<br>• Acceptance test report<br>• Software product<br>• Joint review reports<br>• Traceability of acceptance tests to the requirements baseline | • Software Verification Cases and Procedures<br>• Software Verification Results<br>• Associated Trace Data. |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Portanto, ao compararmos os processos de verificação por testes das normas E-ST-40C e DO-178C, verificamos que a maior diferença está relacionada a abordagem utilizada. A norma espacial apresenta uma abordagem tradicional para os testes seguindo etapas como testes de unidade, integração de

componentes e integração de hardware / software, enquanto que a norma aeronáutica apresenta uma inversão que seguem a sequência: testes de integração hardware / software, integração de componentes e testes de unidade.

Sobre os testes de integração de hardware / software, podemos identificar que os testes acontecem no escopo de sistemas, ou seja, são testes de sistemas que verificam na sua maioria itens relacionados com o sistema e não com o software. Isto pode ser identificado através dos erros que são tipicamente encontrados neste tipo de teste e que são citados pela norma DO-178C.

Podemos observar também que a norma espacial divide os testes em três processos enquanto que a norma aeronáutica mantém os testes organizados em um único processo além de garantir melhor suporte com informações e objetivos claros sobre os testes em cada nível. Esta organização dos processos também se reflete nos artefatos de saída pois, enquanto a norma espacial mantém um número significativo de artefatos, sendo doze somente para o processo de testes, a norma aeronáutica devido à centralização dos processos, possui um número bem reduzido de artefatos de saída, sendo apenas três artefatos que são atualizados ao longo das atividades dos processos.

A norma espacial mantém uma grande interação com o cliente ao final do processo de teste e, consequente, entrega do software. A norma aeronáutica também chama a atenção pelo rigor apresentado nas análises de cobertura, de requisitos e estrutural, refletindo o rigor e a preocupação quanto aos erros de software e seu impacto na certificação do sistema.

Finalmente, conforme observamos nesta comparação, as normas possuem um equilibrado conjunto de processos, a norma espacial apresenta seu conteúdo seis processos relacionados com o desenvolvimento de software e outros três que são cobertos por outras normas do conjunto, deste modo, garantindo mais aprofundamento técnico, enquanto a norma aeronáutica apresenta em seu conteúdo nove processos relacionados com o desenvolvimento de software.

Quanto a estrutura, ambas apresentam uma estrutura de processos semelhantes; entretanto, em uma avaliação mais detalhada destes processos, podemos identificar que alguns destes são bem distintos, como por exemplo, os processos de verificação por testes, no qual a norma DO-178C possui um processo mais rigoroso, chegando inclusive a guiar os testes em código desativado. Diferenças como esta, denotam dois aspectos: o contexto de atuação de cada uma das normas e, principalmente, o amadurecimento dos padrões, também em decorrência do rigor aplicado nos setores.

Nas seções seguintes, apresentaremos uma proposta de processo de verificação tomando por base a comparação supracitada e o que há de melhor nas duas normas.

# 4 PROCESSO DE VERIFICAÇÃO POR TESTES BASEADO NA COMPARAÇÂO DAS NORMAS ECSS-E-ST-40C E RTCA-DO-178C

Este trabalho tem por objetivo propor um processo de verificação por testes baseado na comparação das normas ECSS-E-ST-40C e RTCA-DO-178C.

Logo, este capítulo apresenta o processo de verificação por testes segundo a abordagem mais adotada e praticada pela indústria de software no âmbito embarcado e na sequência, o processo proposto a ser usado no desenvolvimento de softwares embarcáveis. Como um produto do estudo das normas do meio aeroespacial, o processo de verificação por testes apresenta métodos e atividades básicas para seu cumprimento.

## 4.1. Visão Geral do Processo Praticado

A Engenharia de Software evoluiu significativamente nas últimas décadas, estabelecendo novas abordagens, métodos e ferramentas que auxiliam no ciclo de desenvolvimento, desde o levantamento de requisitos até a manutenção do software.

Aliado à necessidade de novas aplicações, isto culminou na utilização de software e sistemas baseados em software em todas as áreas da atividade humana e, em especial, no segmento aeroespacial.

Softwares para aplicações aeroespaciais necessitam ter sua qualidade garantida, logo seu desenvolvimento segue normas e passam por um rigoroso processo de verificação.

De acordo com Blackburn (2004), o aumento da complexidade desses sistemas impactou no processo de verificação e no aumento de seu custo, embora exista um significativo esforço para reduzir esse custo com base no aprimoramento de processos e no avanço tecnológico das ferramentas.

No âmbito de software também há uma evolução nas ferramentas que apoiam testes chegando até a os executar automaticamente; mas, como apontado por

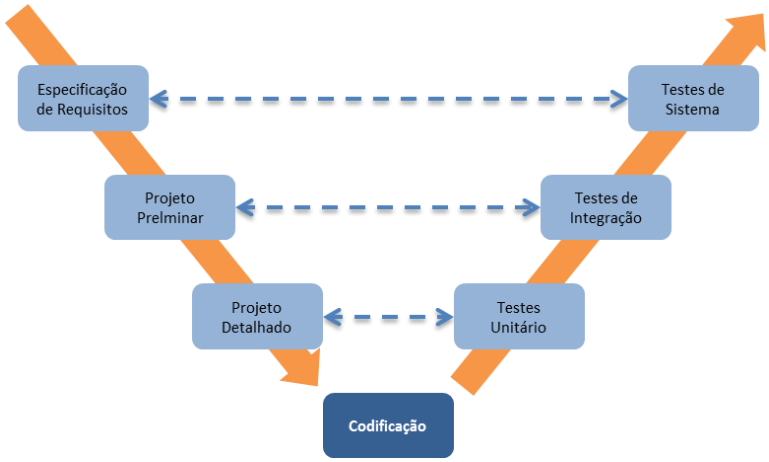
BlackBurn (2004), apesar da automação, esses testes ainda precisam ser criados manualmente.

Dito isto e considerando o contexto de sistemas críticos, softwares embarcados demandam um número elevado de testes, podendo conter centenas ou até milhares de casos de testes. Esses, por sua vez, também são extensos, possuem muitas variáveis de entrada / saída e geram um alto volume de dados.

Uma consideração de substancial importância é a definição de uma abordagem para a realização dos testes, isto impacta no planejamento, na efetiva arquitetura do conjunto de casos de testes e, ainda, na maneira que pequenos componentes serão integrados.

Segundo Myers et al. (2012), a abordagem empregada no processo de testes implica a definição dos casos de teste, a ferramenta que deve ser usada, a ordem em que os componentes são testados e, ainda, o custo de localizar e reparar os erros.

Figura 4.1 – Diagrama do modelo em V.




Fonte: Adaptado de Jorgensen (2002).

A priori, abordagens de testes dividem os esforços e também refletem o ciclo de vida de desenvolvimento utilizado. Um modelo de processo de

desenvolvimento aplicado atualmente e bastante difundido no desenvolvimento de sistemas e software críticos é o modelo em V, como mostra a Figura 4.1.

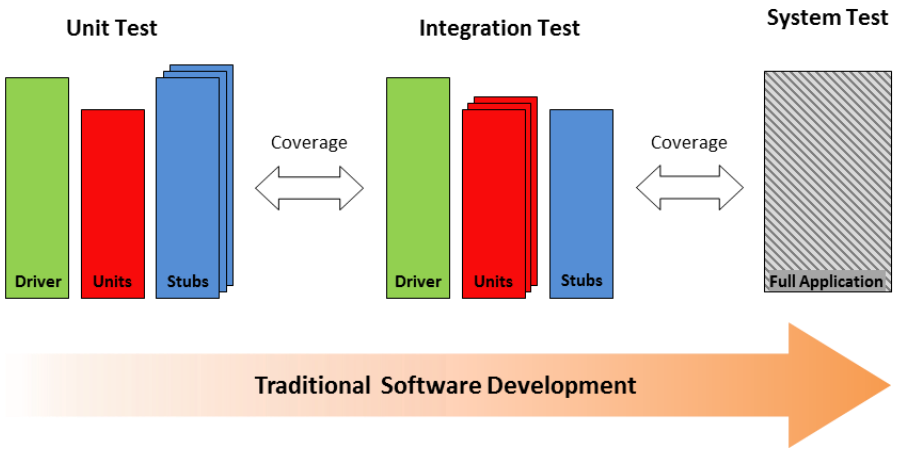
O modelo em V representa as atividades do ciclo de vida do desenvolvimento de um software, desde a especificação de requisitos até a integração de sistema, com variantes que podem representar até a fase de manutenção. O modelo é considerado como uma extensão do modelo cascata, apresentando uma abordagem na fase de testes chamada *Bottom-Up*.

Nesta abordagem, são testados em um primeiro momento os itens de mais baixo nível e o foco das atividades de teste caminha na hierarquia estrutural do sistema até alcançar os componentes de mais alto nível.

Portanto, após todas as atividades que precedem os testes, como planejamento e desenvolvimento de *stubs* e *drivers*, a aplicação das técnicas de teste inicia-se nas unidades de software. Na sequência, são aplicados os testes de integração dos componentes de software e findam-se nos testes de sistema ou funcionais.

Contudo, o modelo em V apresenta algumas desvantagens como aponta vector software (2014): alta demanda de esforço aplicado ao desenvolvimento de *drivers* para testes, risco de desenvolvimento de casos de testes baseados no código desenvolvido, e a identificação tardia de potenciais defeitos no software.

Figura 4.2 – Desenvolvimento tradicional de software.




Fonte: Adaptado de Vector Software (2014).

A Figura 4.2 apresenta uma visão diferente das atividades de teste presentes no modelo em V e, também, ilustra como o processo de verificação por testes avança no escopo do produto de software como também chama atenção para a cobertura de código durante os testes.

Entre as normas citadas neste trabalho, segundo a E-ST-40C, as normas do conjunto ECSS não impõem a adoção de um modelo de ciclo de vida de desenvolvimento específico. Entretanto, o encadeamento dos processos de Engenharia de Software e revisões formais (ver Figura 2.3 na seção 2.6.2 ) naturalmente remetem à adoção do modelo cascata.

Podemos concluir que estas normas do conjunto ECSS preconizam a prática e a adoção um ciclo de vida similar ao modelo cascata, e também que seu processo de verificação segue a abordagem *Bottom-Up* para os testes, ainda que explicitamente não os façam na norma, como habitualmente acontece no meio aeroespacial.

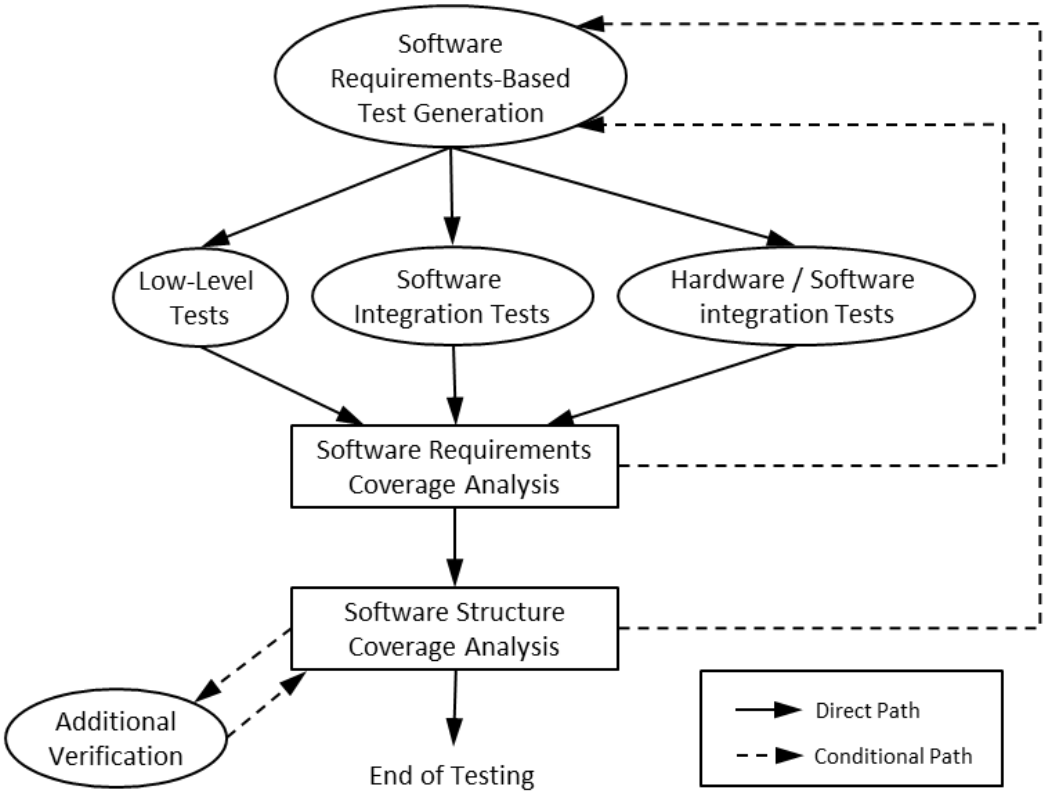
A norma do setor aeronáutico DO-178B, versão que precede a atual DO-178C, propunha o uso de um ciclo de vida de desenvolvimento iterativo devido ao conceito praticado de desenvolvimento incremental das funções de sistema, complexidade e desenvolvimento de requisitos.

A Figura 4.3 ilustra o processo de verificação desta norma apresentando as atividades existentes, na qual é possível identificar a sugestão de uma abordagem *Bottom-Up*, com a aplicação dos testes iniciando a partir dos testes de baixo nível (*Low-Level Tests*), continuando com os Testes de Integração de Software (*Software Integration Tests*) e finalizando com os Testes de integração de Hardware/Software (*Hardware/Software Integration Tests*), embora não exista claramente em seu conteúdo um viés sobre a abordagem a ser praticada, diferentemente de como se apresenta este mesmo processo em sua atual versão (DO-178C).

Myers et al. (2012) menciona que a grande quantidade de unidades e componentes de software implicam na demanda de um grande número de

*drives* para a execução dos testes, apesar da baixa complexidade e facilidade de criação de cada um destes.

Figura 4.3 – Processo de verificação por testes segundo DO-178B.




Fonte: Adaptado de RTCA-DO-178B (1992).

Ainda sobre os testes em unidades de software, segundo Vector Software (2014), para criar uma suíte de testes com alcance e cobertura de código perto de 100% seria necessário gerar uma linha de código de teste (drivers, stubs e dados de teste) para cada linha de código do software a ser testado.

No entanto, como já mencionado, ferramentas computacionais minimizam uma considerável parte do esforço envolvido na preparação e, principalmente, aplicação de um efetivo conjunto de testes. Blackburn (1999) também acrescenta que tais ferramentas são valiosas para reduzir a prevenção de erros manuais no processo de teste, enquanto libera os desenvolvedores a se concentrar na tarefa mais complexa para o desenvolvimento, especificação e análise.

Contudo, verificamos que o desempenho das atividades de teste e a efetividade do processo de verificação demandam o auxílio de ferramentas semiautomáticas.

Quanto às abordagens dos processos de verificação apresentados nesta seção, eles ilustram os processos praticados pela indústria de software, especialmente, a indústria espacial através do conjunto ECSS.

Porém, verificamos que as normas desse conjunto, aqui representada pela norma E-ST-40C, no que tange aos processos de desenvolvimento de software, possuem uma desatualização ante, por exemplo, a norma aeronáutica DO-178C. Isto ocorre pois não tratam aspectos como prática de novas abordagens e incorporação de novas tecnologias de desenvolvimento, quanto a qualificação de ferramentas abordado na norma aeronáutica e ausente na espacial, isto justificativa a ausência da certificação como um exigência.

De acordo com Blackburn (1998), o mercado impulsiona as companhias a definirem ou/e adotarem novas abordagens para reduzir custos, contribuindo também para a melhoria dos processos, principalmente, quando buscamos entregar produtos de qualidade ou galgamos uma certificação.

## 4.2. Visão Geral do Processo Proposto

Atualmente, o cenário das organizações impõe que se faça mais com os mínimos recursos e, para enfrentar o desafio de produzir produtos de qualidade, o uso de ferramentas de automação surge como a chave para agilizar o desenvolvimento sem produzir impactos, como a introdução de defeitos no desenvolvimento.

No que tange ao desenvolvimento de sistemas críticos, a fase de verificação pode demandar uma grande parte do esforço de desenvolvimento, podendo haver um acréscimo quando se busca uma certificação.

Conforme mencionado por Blackburn (2005):

> *Like many companies that build high assurance, life critical applications, zero defects is a requirement. The cost of the verification and validation (V&V) efforts for these companies often exceed fifty percent of the total effort, and the company discussed in this case study did confirm that its testing cost was significantly higher than fifty percent of the life cycle cost (BLACKBURN, 2005).*

A automação das atividades de testes é uma alternativa ao alto custo e esforço alocado. Conforme apontado por Blackburn (1998), o uso de desenvolvimento baseado em modelos e geração automática de testes pode ser uma alternativa para eliminar a maioria das tradicionais atividades de testes. Esta combinação pode reduzir significativamente o tempo e custo do processo de verificação, melhorando o uso dos recursos disponíveis nas organizações.

O desenvolvimento baseado em modelos e a geração automática de casos de teste são alternativas para a redução do trabalho manual e automação de algumas atividades com apoio de ferramentas disponíveis atualmente no mercado. Abordagens como essas são orientadas à redução e a identificação prematura de defeitos, removendo-os antes mesmo de o código ser produzido.

De acordo com Blackburn (2001), a prevenção de defeitos envolve encontrar e corrigir os problemas antes que eles se propaguem ao longo do ciclo de desenvolvimento, sendo mais efetivo durante a fase de requisitos onde o custo de correção dos defeitos encontrados é menor que nas fases posteriores.

Normas de desenvolvimento como a DO-178C guiam quanto ao uso dessas novas tecnologias através de seus suplementos (ver seção 0). Dado à relevância desses suplementos, dois deles merecem destaque: DO-330 – Qualificação de Ferramenta (*Tool Qualification*) e DO-331 - Desenvolvimento Baseado em Modelos (*Model Base Development*).

Segundo a norma DO-330, ferramentas necessitam ser qualificadas quando eliminam, reduzem ou automatizam processos sem uma verificação manual. Logo, quando mencionamos a automação de atividades seja na geração de casos de teste ou atividades do processo de verificação, estamos fazendo uso de ferramentas que se encaixam em duas categorias: ferramentas de desenvolvimento e ferramentas de verificação.

A preocupação com as ferramentas parte da premissa que as próprias ferramentas podem inserir erros quando utilizadas no desenvolvimento, ou pode falhar na detecção de um erro quando o contexto é o de ferramentas de verificação. Este processo de qualificação possui um custo elevado. Embora as ferramentas usadas no desenvolvimento de sistemas críticos possuam um alto custo, os benefícios de sua utilização compensam seu custo, principalmente, quando se visa uma certificação.

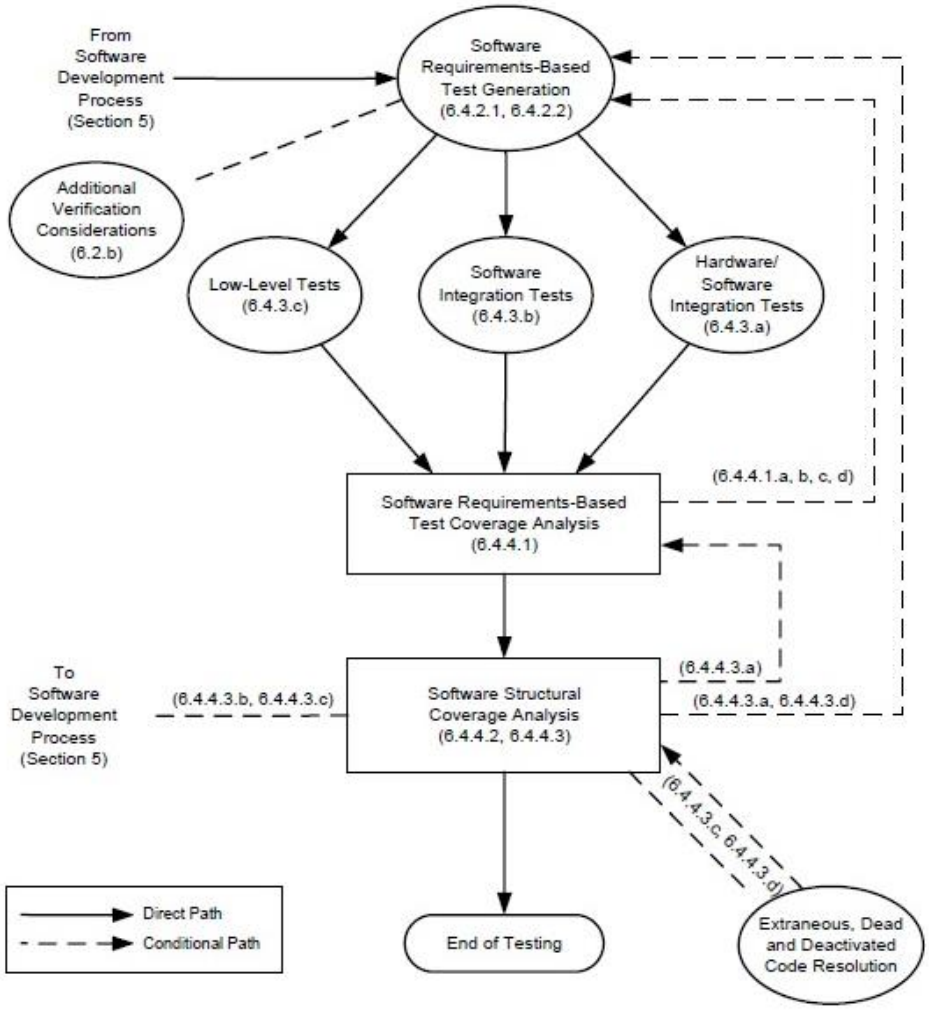
Quanto ao desenvolvimento baseado em modelos, a norma DO-331 provê uma diretriz para orientar o uso desta tecnologia no desenvolvimento de softwares aeronáuticos de forma complementar à DO-178C. Também, encontrado em outras literaturas como Engenharia Dirigida a Modelos (*Model-Driven Engineering - MDE*), esta faz uso de modelos como representações abstrata de aspectos de um sistema ou software, o qual auxilia no processo de desenvolvimento e permite aos desenvolvedores pensar em alto nível e não se preocupar com detalhes de implementação.

O desenvolvimento baseado em modelos é aplicado em uma grande gama de domínios da engenharia como sistemas, softwares, eletrônica, mecânica. A modelagem é realizada através do uso de notação para modelagem, podendo ser gráfica ou textual (SysML, DoDAF, UML, etc).

Além de permitir um alto nível de abstração, outros benefícios alcançados com o uso de modelos é a geração automática de código, geração de testes, suporte à verificação de requisitos e arquitetura. Para Blackburn (1998), o uso de modelos e suas ferramentas possibilitam várias visões do sistema ou software em desenvolvimento.

Alguns trabalhos da literatura apontam que esta abordagem é utilizada com sucesso no desenvolvimento de grandes e complexos sistemas e softwares críticos, como mostram Blackburn et al. (1998), Blackburn et al. (2005) e Wehrmeister et al. (2012).

Figura 4.4 – Processo de verificação conforme DO-178C.




Fonte: Adaptado de RTCA-DO-178C (2012).

No contexto aeronáutico, embora a norma DO-178C disponibilize um complemento para guiar o uso do desenvolvimento baseado em modelos, esta não preconiza e nem aborda diretamente o uso da tecnologia ao longo de seu conteúdo. Sugere, deste modo, que a adoção desta tecnologia é opcional, embora altamente recomendada devido aos benefícios frente uma certificação

e, principalmente, o histórico de sucesso de grandes projetos que fizeram uso da tecnologia conforme mencionado acima.

Como objeto deste trabalho, na sequência vamos abordar com mais detalhes o processo de verificação por testes da norma DO-178C, citado acima (ver seção 0. Tabela 3.9 – Entradas e saídas do processo de verificação na fase de integração.

| **E-ST-40C** | **DO-178C** |
|---|---|
| Verification os Software Integration | Reviews and Analyses of the Outputs of the Integration Process |
| **Entradas** | **Entradas** |
| • Software Component Design Documents and Code<br>• Software Integration Test Plan | • The Object Code<br>• Executable Object Code<br>• Parameter Data Item<br>• Compiling Data<br>• Linking Data<br>• Loading Data |
| **Saídas** | **Saídas** |
| • Software integration verification report | • Software Verification Cases and Procedures<br>• Software Verification Results<br>• Associated Trace Data. |

Fonte: ECSS-E-ST-40C (2009) e RTCA-DO-178C (2012).

Verificação por Testes) e ilustrado através da Figura 4.4, a qual apresenta as atividades do processo que devem ser usadas para atingir os objetivos dos testes de software. Como mostra o diagrama, a norma apresenta três tipos de testes:

- Testes de Integração de Hardware e Software (*Hardware/software integration testing*) – Verificam a correta operação do software no ambiente computacional definido como alvo (*Target*).

- Testes de Integração de Software (*Software integration testing*) – Verifica a relação entre os requisitos de software e os componentes,

além de verificar a implementação dos requisitos de software e componentes de software de acordo com a arquitetura.

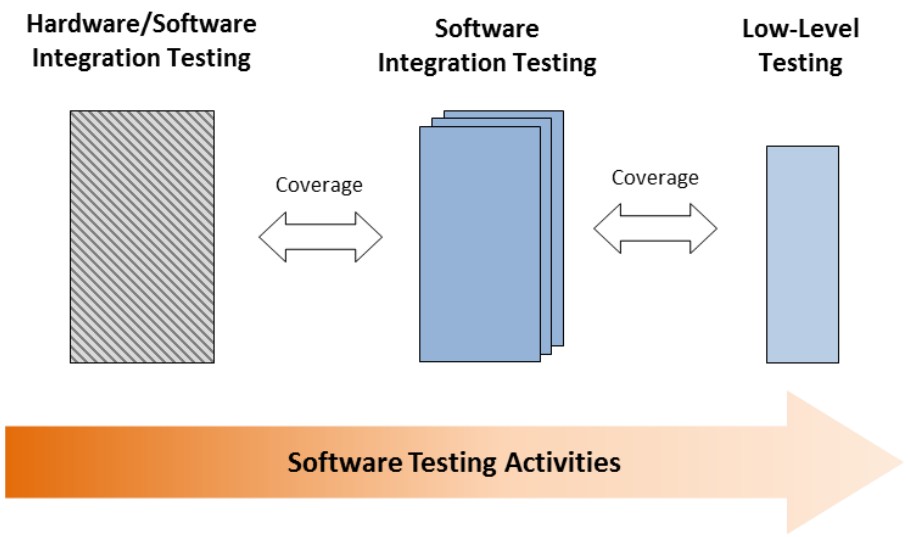
- Testes de Baixo Nível ou unitários (L*ow-level testing*) – verifica a implementação dos requisitos de baixo nível. Basicamente busca erros em itens do código como, por exemplo, lógicas de decisão.

Ainda de acordo com a Figura 4.4, a atividade Geração de Testes Baseado nos Requisitos de Software (*Software Requirements-Based Test Generation*) atua como uma interface entre os processos de desenvolvimento e as atividades de testes de software. Esta atividade é responsável por gerar os casos de teste com base nos requisitos de software. Segundo a norma, esta estratégia de gerar os casos de teste com base nos requisitos tem se mostrado muito efetiva na descoberta de erros.

A norma também orienta para o desenvolvimento de casos de teste segundo dois tipos: Classe Válida (*Normal Range*) e Classe inválida (*Robustness*). O primeiro tipo habilita o software a responder a entradas e condições normais; e o segundo, a entradas e condições anormais. Vale salientar que o desenvolvimento dos casos de testes pode acontecer através da geração automática dos casos de teste com o uso de modelos ou através da geração manual dos casos de teste.

Podemos identificar que a norma apresenta uma abordagem de testes inversa daquelas mencionadas acima (ver seção 4.1). Embora os diagramas que ilustram as atividades de teste das normas DO-178B e DO-178C sejam similares, cabe mencionar que a atual versão (Figura 4.4) ao apresentar em seu conteúdo a inversão das atividades e a inscrição da numeração (6.4.3.a, 6.4.3.b e 6.4.3.c) na representação das atividades no diagrama, sugere que estas atividades de teste sigam a seguinte sequência: Testes de Integração de Hardware / Software, Testes de Integração de Software e Testes de Baixo Nível, como pode ser visto na ilustração da Figura 4.5. Uma descrição detalhada destes três níveis de testes será apresentada mais abaixo.

Figura 4.5 – Atividades de teste de software.




Fonte: Produção do autor.

Um aspecto importante a ser destacado e que justifica a inversão da abordagem de testes utilizada no processo é o uso de um ambiente simulado ou emulado. Sabe-se que em projetos de desenvolvimento de sistemas os cronogramas de desenvolvimento de software e hardware não caminham em paralelo e este artificio de utilizar um ambiente simulado ou emulado permite adiantar os testes do software embarcado, permitindo que erros detectados somente no momento de integração de hardware / software sejam encontrados o quanto antes no processo de desenvolvimento. Portanto, se ganha em termos de prazo, uma vez que na tradicional abordagem seria necessário aguardar o desenvolvimento do hardware para realizar tais testes.

Os testes verificam a implementação dos requisitos de software. Como parte integrante do processo de verificação e das atividades de testes, a análise de cobertura determina o quão efetivo foram os testes realizados para cobrir os requisitos e a estrutura do software. Esta análise dos testes é realizada através de dois passos: Análise de Cobertura Baseada em Requisitos (*Requirements-Based Coverage Analysis*) e Análise de Cobertura Estrutural (*Structural Coverage Analysis*).

Neste processo, a Análise de Cobertura Baseada em Requisitos determina o quanto os testes foram efetivos para verificar a implementação dos requisitos de software e podem identificar a necessidade de casos de teste adicionais. Já a Análise de Cobertura Estrutural determina qual parte do código não foi exercitada através dos procedimentos de teste, produzindo uma verificação adicional para prover a cobertura estrutural apropriada ao nível de software.

Segundo a observação retirada da norma RTCA DO-178C seção 6.4:

> *If a test case and its corresponding test procedure are developed and executed for hardware/software integration testing or software integration testing and satisfy the requirements-based coverage and structural coverage, it is not necessary to duplicate the test for low-level testing. Substituting nominally equivalent low-level tests for high-level tests may be less effective due to the reduced amount of overall functionality tested (RTCA DO-178C, 2012).*

De acordo com o trecho supracitado retirado da norma DO-178C, este menciona que não há a necessidade de duplicar os procedimentos de teste e respectivos casos de teste desenvolvidos para os testes de integração de hardware / software e integração de software nos testes de baixo nível, embora também mencione que está substituição de testes de baixo nível por testes de alto nível deve ser menos efetiva devido a quantidade de funcionalidades testadas e, ainda, sugere que se pode haver dificuldades de execução dos testes.

Alternativamente, pode se haver uma redução dos testes de baixo nível se os casos de testes desenvolvidos e aplicados aos níveis superiores (Integração Hardware / Software e Integração de Software) atenderem aos critérios para a cobertura para os requisitos.

Contudo, acerca dos testes de software segundo a norma DO-178C, há dois aspectos importantes a serem observados:

1) Com o intuito em satisfazer alguns dos objetivos dos testes de software, busca-se testar o software no ambiente, ou seja, carregar o software em um ambiente que apresente o comportamento próximo do ambiente computacional da instalação final. Entretanto, o órgão certificador pode creditar testes feitos em ambientes emulados e, também, ambientes simulados em computador, porém, a norma destaca que testes selecionados devem fazer uso do ambiente físico (*target*) considerando que, alguns erros somente são detectáveis neste ambiente.

2) Como aponta a norma, a cobertura dos requisitos e a cobertura estrutural do código deve ser alcançada com um controle e monitoramento preciso das entradas e execução do código, sendo isto possível, geralmente, em um ambiente de testes totalmente integrado (*Hardware-in-the-loop*) e disposto de ferramentas, as quais, quando se busca uma certificação, devem ser qualificadas.

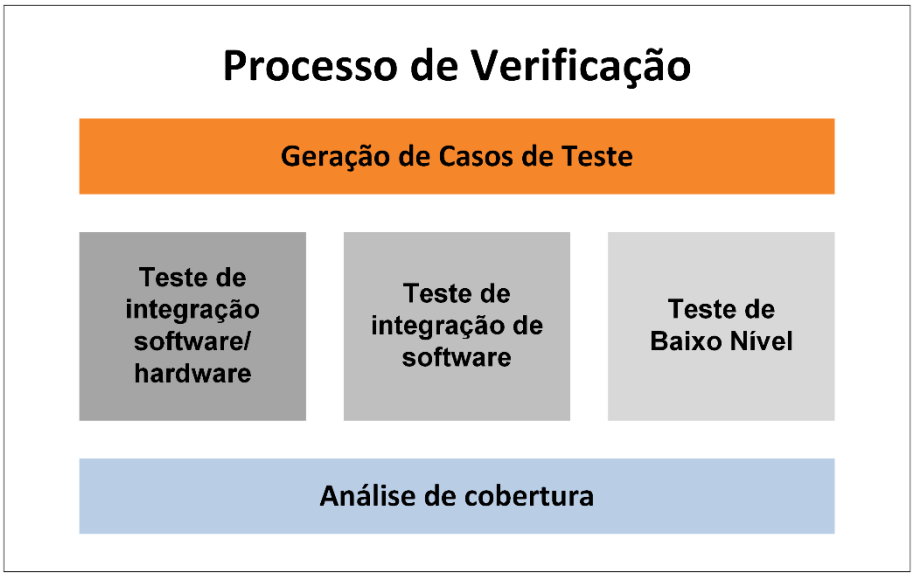
### 4.3. Visão Detalhada do Processo Proposto

A norma aeronáutica DO-178C vem sendo adotada pela indústria de software crítico de outros segmentos industriais como uma alternativa para a produção de software seguro, devido ao seu rigor e objetividade.

Motivo pelo qual também adotamos esta norma como base para desenvolvermos uma proposta de processo de verificação por testes, que seja simples e eficiente na busca por erros no software e que atenda à demanda de pequenos projetos e empresas no setor aeroespacial.

O processo propõe procedimentos básicos para a realização dos testes de software conforme três níveis: Testes de Integração de Hardware / Software, Testes de Integração de Software e Testes de Baixo Nível, conforme ilustrado na Figura 4.6. Também são propostas atividades para seleção e geração de casos de testes com base nos requisitos de software e, ainda, atividades para verificar a eficácia dos testes executados no que tange à cobertura de requisitos e cobertura estrutural do código fonte.

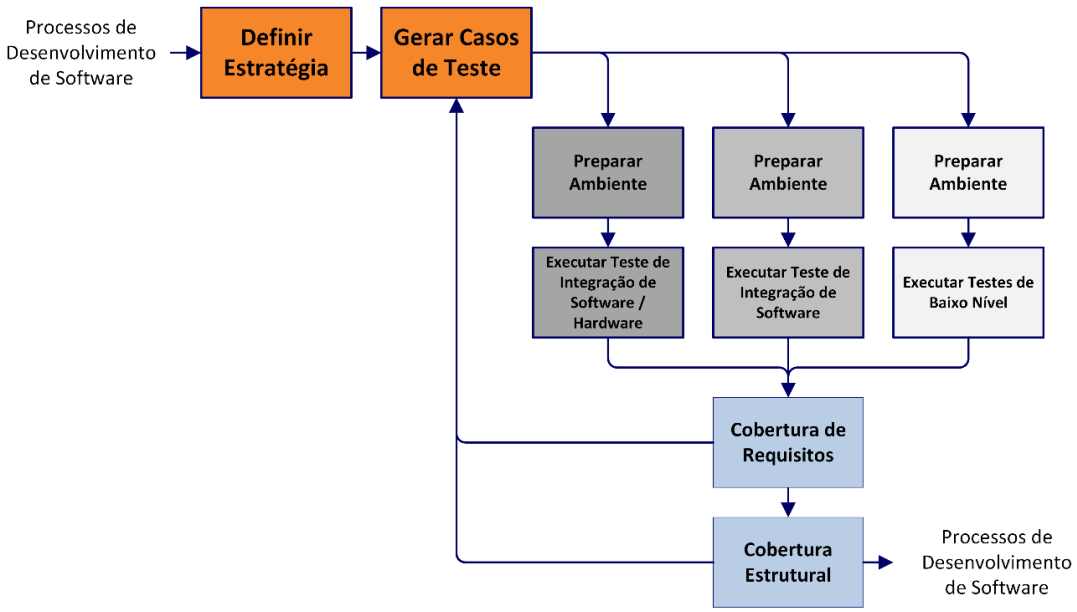
Figura 4.6 – Diagrama do processo proposto.




Fonte: Produção do autor.

O processo de verificação de software faz interface com os demais processos do ciclo de vida de desenvolvimento. O subprocesso Geração de Casos de Teste é a interface de entrada, e o subprocesso Análise de Cobertura é interface de saída, trocando informações com os demais processos. Ambos são ilustrados pela Figura 4.7 e detalhados adiante.

Figura 4.7 – Visão geral das atividades do processo proposto.




Fonte: Produção do autor.

Na sequência iremos estabelecer e detalhar o processo proposto, através dos subprocessos e suas respectivas atividades, identificando segundo a modelagem IDF0 (NIST, 1993), as entradas, saídas, controles e mecanismos básicos para a execução do processo.

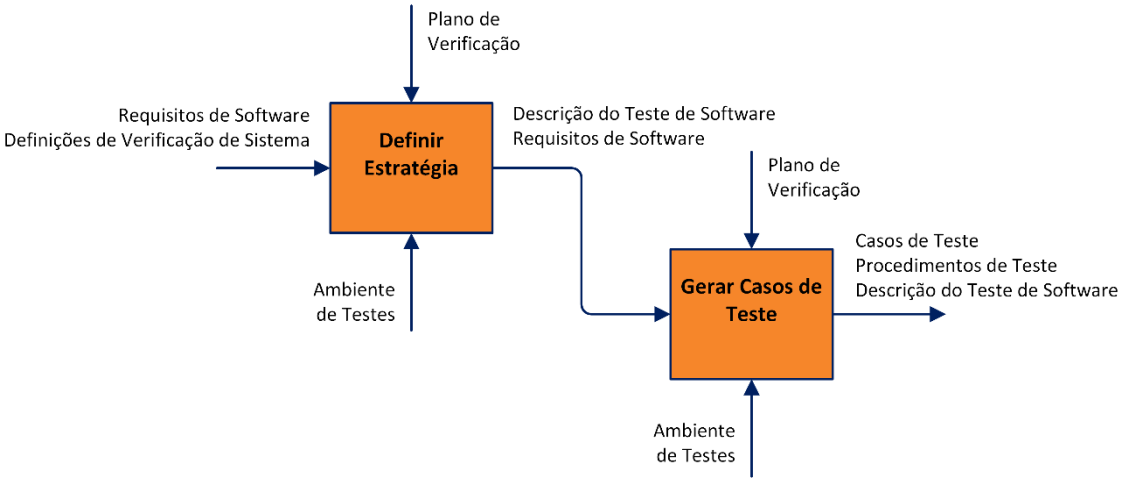
### 4.3.1. Geração de Casos de Teste

O propósito do subprocesso **Geração de Casos de Teste** é desenvolver e documentar os procedimentos de teste e seus respectivos casos de teste para serem utilizados na realização dos testes de software.

Alternativamente, o início das atividades deste subprocesso pode se dar juntamente com o levantamento dos requisitos na avaliação da *baseline* de requisitos, atuando para uma melhor qualidade dos requisitos e, consequentemente, dos casos de teste gerados a partir destes.

O diagrama da Figura 4.8 identifica as atividades pertencentes ao subprocesso além das entradas, controles, mecanismos e saídas.

Figura 4.8 – Modelo IDF0 do subprocesso geração de casos de teste.




Fonte: Produção do autor.

Conforme citado acima, o conjunto de casos de teste pode ser desenvolvido manualmente ou com o auxílio de modelos. Esta proposta de processo prioriza a geração manual dos casos de teste, devido à demanda de ferramentas e de

recursos especialistas que este envolve, embora fortemente se recomende o uso de modelos para diminuir o esforço envolvido na atividade.

De acordo com a avaliação de cobertura realizada pelas atividades do subprocesso **Análise de Cobertura**, novos procedimentos de testes e casos de testes podem ser desenvolvidos. Esta iteração acontece até que um nível de cobertura dos requisitos e de estrutura de código seja satisfeito. Entretanto, a cada iteração um dos três tipos de teste poderá ser realizado.

#### 4.3.1.1. Definir Estratégia

O propósito desta atividade é avaliar as entradas (requisitos de software e as informações sobre a verificação de sistemas que ocorre dentro do escopo do software) e definir a estratégia para desenvolver os casos de testes, procedimentos de testes e a execução dos testes. Isto inclui: documentar o propósito de cada caso de teste, conjunto de entradas, condições, resultados esperados e os critérios de aprovação/falha. Detalhar passo-a-passo as configurações dos casos de teste e as informações necessárias (variáveis utilizadas, entradas, saídas, precedência de casos ou procedimentos de teste anteriores) para sua implementação na ferramenta e execução, avaliação do resultado dos testes e a configuração de ambiente a ser usado.

#### 4.3.1.2. Gerar Casos de Testes

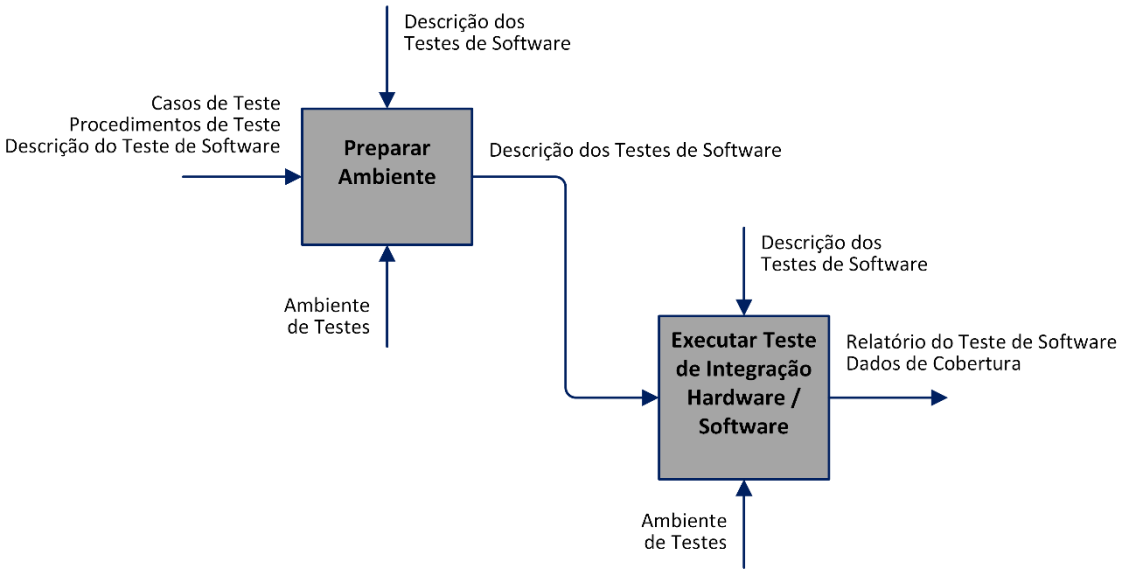
O propósito desta atividade é gerar os casos de teste e os procedimentos no ambiente de teste adotado de acordo com a estratégia e documentação definida. A entrada para a atividade são os requisitos de software e uma primeira versão do documento de descrição dos testes. Na saída da atividade o documento de descrição dos testes sofre atualização para embasar os testes de software nas atividades seguintes.

### 4.3.2. Teste de Integração de Hardware/Software

A Figura 4.9 apresenta o modelo IDF0 do subprocesso Teste de Integração de Hardware / Software e suas atividades que serão descritas abaixo.

O propósito do suprocesso **Teste de Integração Software/Hardware** é preparar e executar os testes de integração de hardware/software, concentrando nos erros oriundos da operação do software embarcado no ambiente computacional escolhido (*target*) e de acordo com as funcionalidades de alto nível.

Figura 4.9 – Modelo IDF0 do subprocesso teste de integração de HW/SW.




Fonte: Produção do autor.

### 4.3.2.1. Preparar Ambiente para teste de hardware / software

O propósito desta atividade é preparar o ambiente para os testes de integração de hardware / software, o que envolve preparar o software a ser testado, o ambiente do hardware físico, simulado ou emulado além do ambiente de testes. Instalação de softwares adicionais, arquivos e bibliotecas utilizadas durantes os testes. Preparação das ferramentas de testes e arquivos de suporte. Isto também inclui computadores, interfaces e outros dispositivos.

A atividade recebe os casos de teste, procedimentos de teste e o documento de descrição dos testes para orientar a preparação dos testes. O documento de descrição dos testes segue como um guia para a execução dos testes. A Tabela 4.11 apresenta um exemplo de um procedimento de teste contendo o método utilizado, dados de entrada, saída esperada e resultado do teste.

Tabela 4.11 – Exemplo de descrição de um procedimento de teste

| Caso de Teste | Procedimento | Variável | Valor Entrada | Valor Saída | Valor Atual | Status |
|---|---|---|---|---|---|---|
| PT01_01 | getPower | us_Power | | | | |
| PT01_02 | sm_run | b_On | | | | |
| | | us_State | | | | |
| PT01_03 | sm_run | b_On | | | | |
| | | b_Fail | | | | |
| | | us_State | | | | |
| Resultado | Statement Coverage – (Porcentagem de cobertura)<br>Branch Coverage – (Porcentagem de cobertura)<br>MC/DC Coverage – (Porcentagem de cobertura) | | | | | |

Fonte: Produção do autor.

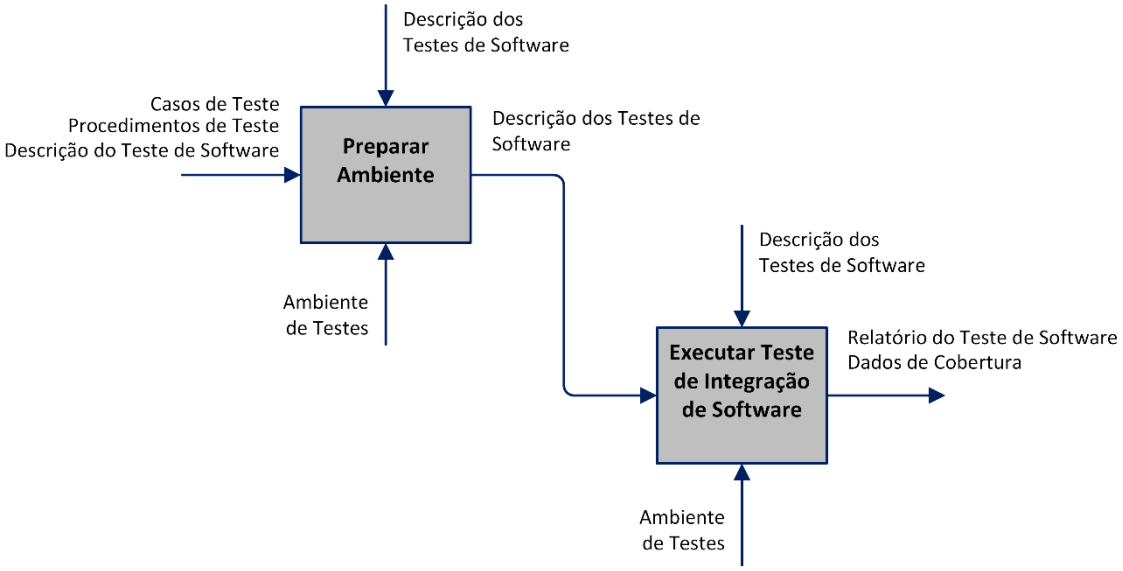
#### 4.3.2.2. Executar os testes de hardware / software

O propósito da atividade é executar os testes de integração de hardware / software de acordo com o documento de descrição dos testes. Esta atividade produz como saída um relatório dos testes de software e um arquivo com dados da cobertura estrutural dos testes.

### 4.3.3. Teste de Integração de Software

O propósito do subprocesso **Teste de Integração Software** é preparar e executar os testes de integração de software, concentrando na integração dos módulos de software e suas corretas interações, visando a implementação dos requisitos de software pela arquitetura.

Figura 4.10 – Modelo IDF0 do subprocesso teste de integração de software.




Fonte: Produção do autor.

A Figura 4.10 apresenta o modelo IDF0 do subprocesso Teste de Integração de Software e suas atividades, que serão descritas abaixo.

#### 4.3.3.1. Preparar Ambiente para teste de integração de software

O propósito desta atividade é preparar o ambiente para os testes de integração de software. Neste caso a preparação envolve o software a ser testado e o ambiente de testes, o qual é mencionado adiante. As atividades relacionadas nesta preparação incluem: o carregamento de arquivos adicionais (*drivers* de hardware), bibliotecas utilizadas durante os testes; preparação das ferramentas de teste com as configurações de interface entre as ferramentas que são descritas adiante.

A atividade recebe os casos de teste, procedimentos de teste e o documento de descrição dos testes para orientar a preparação dos testes deste nível. O documento de descrição dos testes deve guiar a atividade de execução dos testes.

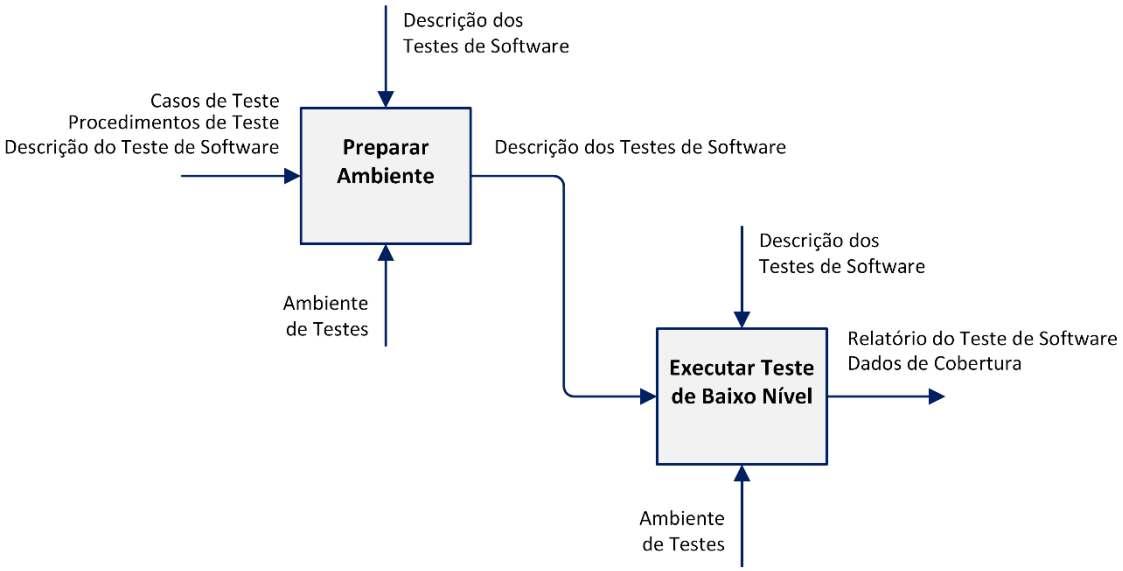
#### 4.3.3.2. Executar os Testes de integração de software

O propósito da atividade é realizar os testes de integração de software de acordo com o documento de descrição dos testes de software recebido na entrada. Produz como saída um relatório dos testes de software e o arquivo com os dados da cobertura de execução dos testes.

### 4.3.4. Teste de Baixo Nível

O propósito do subprocesso **Teste de Baixo Nível** é preparar e executar os testes de baixo nível de software, concentrando em demonstrar que os componentes de baixo nível cumprem os requisitos de baixo nível.

Figura 4.11 – Modelo IDF0 do subprocesso teste de baixo nível.




Fonte: Produção do autor.

A Figura 4.11 apresenta o modelo IDF0 do subprocesso Teste de Integração de Software e suas atividades, que serão descritas abaixo.

#### 4.3.4.1. Preparar Ambiente para teste de baixo nível

O propósito desta atividade é preparar o ambiente para os testes de baixo nível, verificando o carregamento do software a ser testado e o próprio ambiente de testes. Apesar de testes anteriores terem sido realizados, alguma

modificação seja necessária após a execução dos testes anteriores ou de nível superior, o que pode incluir: o carregamento de novos arquivos adicionais (*drivers* de hardware) e novas bibliotecas a serem usadas durante os testes.

A atividade recebe os procedimentos e casos de teste gerados e o documento de descrição dos testes para orientar a preparação para os testes. A atividade produz como saída, o documento de descrição dos testes que deve guiar a execução dos testes.

#### 4.3.4.2. Executar os Testes de baixo nível

O propósito da atividade é realizar os testes de baixo nível de acordo com o documento de descrição dos testes de software. Produz como saída um relatório dos testes de software para este nível de testes e o arquivo com os dados da cobertura de execução dos testes.

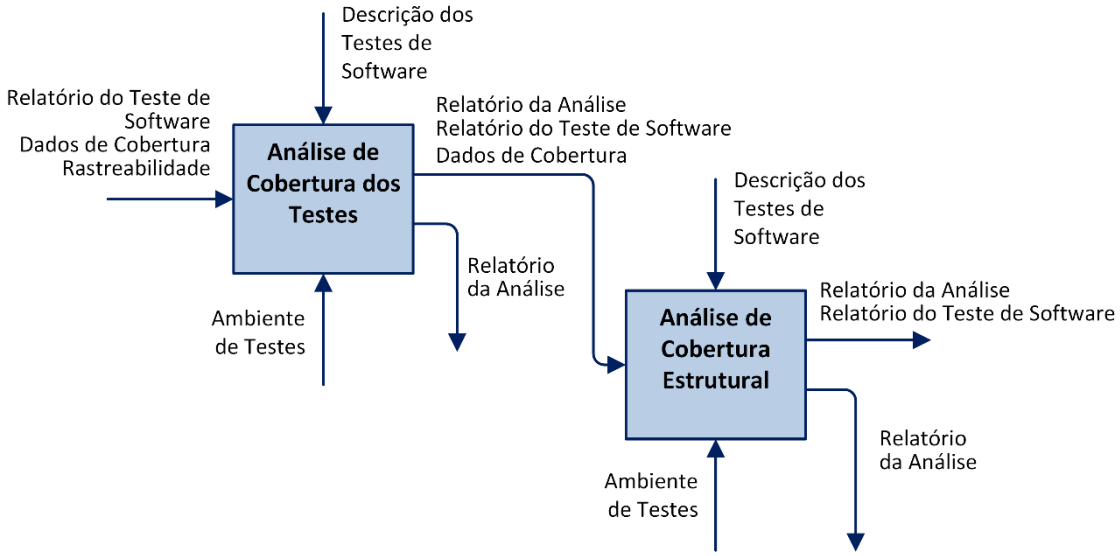
### 4.3.5. Análise de Cobertura

O propósito do subprocesso **Análise de Cobertura** é determinar se os casos de teste selecionados satisfazem os critérios de cobertura relacionada aos requisitos de software e estrutura de código. A análise de cobertura tem por base os objetivos:

- Alcance da cobertura dos requisitos de alto nível.
- Alcance da cobertura dos requisitos de baixo nível.
- Alcance dos critérios apropriados de cobertura da estrutura do software.
- Alcance de cobertura para acoplamento de dado e acoplamento de controle.

A análise é dividida em duas atividades, conforme ilustrado na Figura 4.12 e descrito abaixo. As análises podem identificar que novos casos de testes sejam necessários para alcançar uma maior cobertura, tanto dos requisitos de software quanto da estrutura do código.

Figura 4.12 – Modelo IDF0 do subprocesso de análise de cobertura.




Fonte: Produção do autor.

#### 4.3.5.1. Cobertura dos Requisitos

O propósito desta atividade é determinar através de uma análise se os testes verificaram a implementação dos requisitos de software, na qual se deve garantir um índice de cobertura de 100%. A análise tem como base os itens abaixo:

- Existência de casos de teste para cada requisito de software.
- Caso de teste satisfaz os critérios para os testes do tipo normal e anormal.
- Identificação de deficiências nos casos de teste para melhorias.
- Garantia de que todos os casos de teste e procedimentos são rastreáveis aos requisitos.

A atividade recebe de cada nível de teste o relatório dos testes de software e um arquivo com os dados de cobertura da estrutura de código, e a rastreabilidade dos casos de teste com os requisitos. Caso tenham passado

nos critérios de cobertura, os mesmos artefatos, com exceção da rastreabilidade, seguem para a próxima atividade.

Nos casos em que a cobertura foi insuficiente ou outros problemas foram encontrados nos casos de teste, a atividade emite o relatório da análise, o qual segue para o desenvolvimento de novos casos de teste.

#### 4.3.5.2. Cobertura Estrutural

O propósito desta atividade é determinar qual parte da estrutura do código, incluindo interface entre componentes, não foi exercitada pelos procedimentos de testes baseados nos requisitos. A análise de cobertura estrutural tem por base os seguintes pontos:

- Análise de informação colhida dos testes de cobertura de requisitos, para confirmar que o grau de cobertura é apropriado ao nível do software.
- Análise de código adicional gerado por compilador ou vinculador (*linker*).
- Análise para confirmar que os testes de cobertura de requisitos exercitaram o acoplamento de dados e controle entre os componentes de código.

A atividade recebe, como entrada de cada nível de teste, um arquivo com as informações da cobertura dos testes baseados nos requisitos. No caso de cumprimento dos critérios de cobertura, o relatório de análise é atualizado e emitido como saída da atividade e do subprocesso.

Nos casos em que a cobertura foi insuficiente ou outros problemas foram encontrados nos casos de teste, a atividade emite o relatório da análise, o qual segue para o desenvolvimento de novos casos de teste.

# 5 APLICAÇÃO A UM SOFTWARE ESPACIAL EMBARCÁVEL

Este capítulo apresenta a aplicação do processo proposto neste trabalho, buscando validar a abordagem desenvolvida através da realização de um estudo de caso. O processo foi aplicado a uma máquina de estados desenvolvida pela empresa Homine para aplicação em sistemas críticos e conforme uma customização das normas aeroespaciais com auxílio do grupo de Supervisão de Bordo (SUBORD), dentro da Divisão de Eletrônica Aeroespacial (DEA), da Coordenadoria de Engenharia e Tecnologia Espacial (ETE).

Com este estudo de caso baseado em um produto já testado, busca-se validar o trabalho desenvolvido e avaliar a abordagem proposta quanto ao uso e a contribuição de ferramentas semiautomáticas de teste e, sobretudo, as vantagens e desvantagens desta abordagem onde se propõe a inversão da sequência de testes frente à tradicional abordagem.

O estudo de caso utiliza somente a máquina de estados de um software embarcável, já testado e utilizado, o qual foi desenvolvido segundo um padrão de codificação **MISRA C,** largamente adotado no contexto de sistemas embarcados.

## 5.1. Configuração dos Testes

Um conjunto de testes estimula o software a realizar as mesmas ações que um usuário final ou o próprio sistema poderiam fazer, eventualmente desdobrando em erros a partir de combinações de entradas não desejadas.

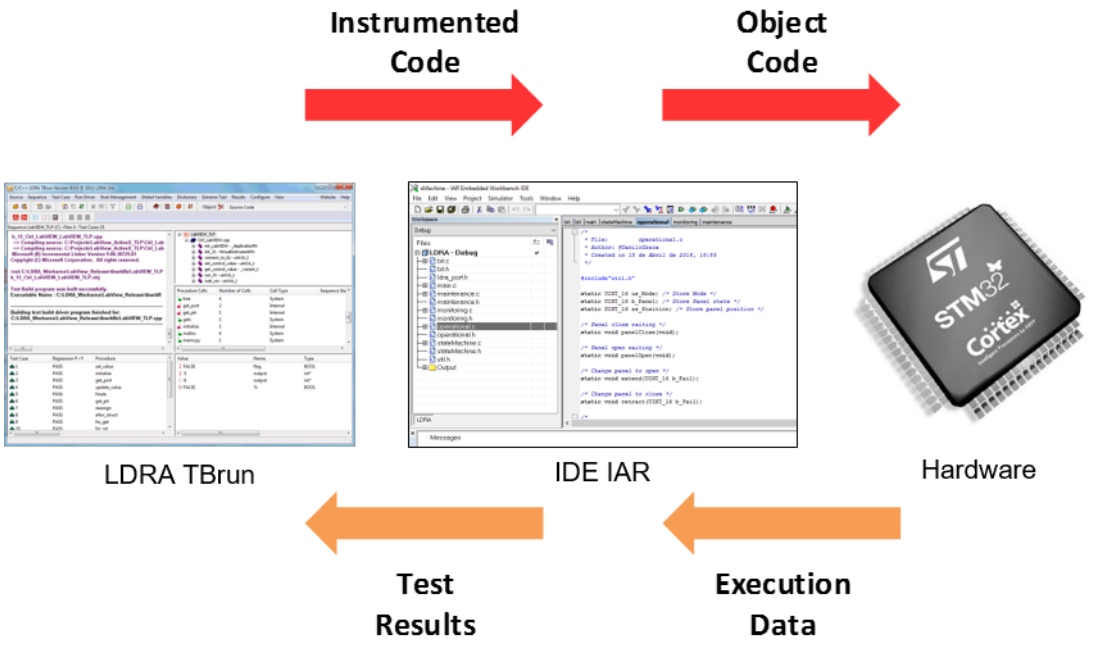
Neste contexto, o experimento busca realizar os testes a partir de um conjunto de casos de testes desenvolvidos manualmente e gerenciados com o suporte de uma ferramenta, ambos descritos abaixo. Conforme a abordagem de testes proposta, os testes serão aplicados de acordo com a sequência: 1) teste de integração de Hardware/Software; 2) teste de integração de software e; 3) testes de baixo nível de software, avaliando a cobertura dos requisitos e a

cobertura estrutural do código fonte a partir da execução dos casos de testes desenvolvidos com base nos requisitos.

Como mencionado anteriormente, ferramentas semiautomáticas de teste contribuem massivamente nas atividades de teste de software suportando e gerindo a quantidade de informações produzidas bem como a automação de algumas atividades, como a gestão dos procedimentos de teste. Com isto, na busca por melhorias no processo, este trabalho fará uso da suíte de testes da LDRA, que permite testes de sistemas e software embarcados e, ainda, a integração com outras ferramentas e ambientes.

Uma das propostas do processo de verificação é testar o software embarcado em um hardware. Para isto o experimento contará com um ambiente computacional na configuração *host / simulation,* no qual a ferramenta TBrun da suíte LDRA é integrada com a IDE IAR e esta, por sua vez, integrada ao hardware simulado. Na qual, através deste ambiente, é possível a comunicação com o hardware a e gestão dos testes a partir da própria ferramenta de testes, conforme ilustrado através da Figura 5.1.

Figura 5.1 – Ambiente de testes.




Fonte: Produção do autor.

O procedimento para testar o software embarcado inicia-se com o projeto da IDE IAR, que contém o código fonte e as configurações sobre o hardware, sendo carregado na ferramenta LDRA TBrun. Nesta ferramenta, configura-se o local de descarregamento do código instrumentado que irá executar no hardware através do IAR. Após estas configurações, a ferramenta envia o código para a IDE IAR que está conectada no hardware (*target*). O software é executado no hardware e um arquivo com o resultado dos testes é disponibilizado para a ferramenta (*Execution Data*). Este arquivo contém informações como quais partes do código foram exercitadas. Estas informações são organizadas na ferramenta LDRA (*Test Results*) e disponibilizadas através de vários gráficos para serem analisados. Conforme a necessidade, novos casos de testes são gerados e o processo ocorre novamente até se atingir a cobertura necessária.

Quanto aos testes utilizados neste experimento, usa-se duas técnicas: testes funcionais e testes estruturais. Justificam-se estas escolhas, uma vez que a técnica funcional avalia a cobertura dos requisitos e a cobertura da estrutura de código com base nos procedimentos de teste desenvolvidos com base nos requisitos, e também, a técnica estrutural possibilita avaliar a estrutura do código fonte, verificando métricas como complexidade e manutenabilidade. Embora testes de estrutura de código fonte não sejam baseados em requisitos e, até mesmo, não sejam creditados em uma certificação, estes testes contribuem direta e efetivamente para a qualidade do produto de software. Conforme mencionado, os testes funcionais e estruturais são suportados e gerenciados pela ferramenta.

Em complemento aos testes supracitados, usa-se também os recursos disponíveis na ferramenta para avaliar o código fonte conforme um padrão de codificação, o qual também é requisitado pelas normas de desenvolvimento de software do setor aeroespacial.

Em resumo, o cenário do experimento segue as seguintes etapas que serão desenvolvidas conforme o processo proposto.

a) Desenvolvimento dos casos de teste: casos de testes serão gerados sem o uso da tecnologia de geração automática de casos de teste, isto será feito com base nos requisitos disponibilizados.

b) Teste de integração hardware/software: serão aplicados testes funcionais, para avaliar a adequação do software quanto aos requisitos de alto nível, verificando seu funcionamento do ponto de vista da entrega de funções, através do estímulo das entradas e coleta das saídas com o software embarcado no hardware. A análise dinâmica também realiza uma avaliação inicial da cobertura rodando no hardware o código fonte.

c) Testes de integração de software: testes funcionais e/ou estruturais serão aplicados para verificar a cobertura dos requisitos pela arquitetura.

d) Testes de baixo nível: testes funcionais e/ou estruturais serão aplicados para verificar a adequação dos requisitos de baixo nível em complemento aos testes de níveis superiores.

e) Avaliação da cobertura: serão avaliados os resultados de testes em cada um dos níveis para determinar se foram suficientes para cobrir os requisitos e a estrutura do código, podendo haver a necessidade de modificação ou criação de novos casos de testes com base nos requisitos.

Um aspecto deve ser observado acerca deste experimento é que aplicamos somente os testes funcionais e estruturais. Isto é devido à grande variedade de tipos de testes e técnicas disponíveis. Sendo assim, adiante iremos descrever os tipos selecionados.

## 5.2. Detalhando a Ferramenta

A suíte de testes LDRA além de testar o software embarcado no hardware, permite a integração com outros ambientes, automação das atividades de testes e a verificação da conformidade com padrões de código. A suíte

apresenta uma completa solução para o desenvolvimento de softwares que vai desde a fase de requisitos até aos testes.

A ferramenta TBrun, utilizada neste estudo de caso, permite a gestão e o armazenamento de casos de teste para reuso. Também permite avaliar a qualidade de código através de revisões estruturais do código, aqui chamada de **análise estática**, enquanto a **análise dinâmica**, executa o código para demonstrar a cobertura de declaração, sendo está um tipo de teste de cobertura estrutural do código fonte.

A Tabela 5.1 identifica os tipos de análise de código realizadas durante a etapa denominada análise estática, as medidas **Clareza, Manutenabilidade e Testabilidade** são compostas por diversas métricas, que foram descritas na seção 2.5 Testes.

Tabela 5.1 – Medidas coletadas na análise estática.

| Análise | Descrição |
|---|---|
| Aderência ao Padrão de Código | Verificar a conformidade do código com relação a um padrão de desenvolvimento garantindo um código mais robusto, seguro e de qualidade. |
| Clareza | Código conciso e de fácil entendimento. |
| Manutenabilidade | Avalia o quanto um software é fácil de ser compreendido e modificado. |
| Testabilidade | Código com alto índice de detecção de falha. |
| Chamada de Função | Verifica se todas as funções possuem uma chamada dentro do código. |

Fonte: Vector Software (2013).

Estas análises contribuem para o aumento da qualidade do produto código, principalmente nos testes de unidades onde a identificação antecipada de erros e defeitos evita os custos desnecessários e onerosos identificados em fases posteriores. Unidades de código sem erros facilitam a integração e os respectivos testes.

Com a análise dinâmica tem-se o índice de cobertura da estrutura do código inicial encontrado através do teste de **Cobertura de Declaração**, que é complementado com as execuções dos procedimentos de teste e, com estes, os demais tipos de cobertura: **Cobertura de Ramo** e **MC/DC**. Estes três tipos de testes relacionados a cobertura da estrutura do código fonte foram detalhados na seção 2.5 - Testes.

Em suma, através da execução dos procedimentos de testes, os requisitos funcionais são verificados e, deste modo, contribuem para determinar o índice de cobertura estrutural do código. Ainda, a ferramenta permite a gestão, armazenamento e reuso de casos de teste e procedimentos de teste, e também suporta, a criação automática de *drivers*, relatórios e controle de mudança.

## 5.3. Descrição do Software Utilizado

O software desenvolvido para o experimento implementa uma máquina de estados para um sistema embarcável crítico e seguro. Este componente de software é um reuso de um componente utilizado em sistemas embarcados desenvolvidos pela Homine para a indústria e utilizado em dois projetos: um sistema de controle de uma turbina de geração de energia e um sistema de uma plataforma inercial para uso aeroespacial.

A Tabela 5.2, apresenta os requisitos utilizados para a implementação desse componente de software, na qual apresenta a identificação do requisito, a descrição e o método de verificação apresentado para cada requisito. Neste contexto, os requisitos que possuem Teste como método serão associados aos seus respectivos casos de testes mais adiante. A Tabela 5.3, apresenta os estados do software e Tabela 5.4, apresenta os modos associado aos estados Operacional e Reduced do software, os quais são relacionados adiante.

Tabela 5.2 – Requisitos do componente de software.

| Requisito | Descrição | Método de Verificação |
|---|---|---|
| SM_REQ01 | O sistemas deve ter um estado STANDBY com modo único. | Análise |
| SM_REQ02 | O sistemas deve ter um estado IBIT com modo único. | Análise |
| SM_REQ03 | O sistemas deve ter um estado READY com modo único. | Análise |
| SM_REQ04 | O sistemas deve ter um estado FAIL com modo único. | Análise |
| SM_REQ05 | O sistemas deve ter um estado EMERGENCY com modo único. | Análise |
| SM_REQ06 | O sistemas deve ter um estado MAINTENANCE com modo único. | Análise |
| SM_REQ07 | O sistemas deve ter um estado OPERATIONAL. | Análise |
| SM_REQ08 | O sistemas deve ter um estado REDUCED. | Análise |
| SM_REQ09 | O estado OPERATIONAL deve apresentar 5 estados (PANEL_OPEN, PANEL_CLOSE, RETRACT, EXTENDED, SHUTDOWN) | Análise |
| SM_REQ10 | O estado REDUCED deve ser um estado virtual associado ao OPERATIONAL | Análise |
| SM_REQ11 | O sistema deve iniciar no estado STANDBY após ser energizado. | Teste |
| SM_REQ12 | O sistema deve permitir trocar do estado STANDBY para o IBIT. | Teste |
| SM_REQ13 | O sistema deve realizar testes de inicialização no estado IBIT. | Análise |
| SM_REQ14 | O sistema deve alterar para estado EMERGENCY se nos testes de inicialização a potência for menor ou igual a 200 W. | Teste |
| SM_REQ15 | O sistema deve alterar para estado FAIL se nos testes de inicialização a potência for menor ou igual a 500 W. | Teste |
| SM_REQ16 | O sistema deve alterar para estado READY se os testes de inicialização não encontrarem falhas (alcançar 500 ciclos). | Teste |
| SM_REQ17 | O sistema deve permitir alterar do estado READY para o estado MAINTENANCE | Teste |
| SM_REQ18 | O sistema deve permitir alterar do estado MAINTENANCE para o estado READY | Teste |



Continua

Tabela 5.2 - Continuação

| Requisito | Descrição | Método de Verificação |
|---|---|---|
| SM_REQ19 | O sistema deve permitir alterar do estado READY para o estado OPERATIONAL | Teste |
| SM_REQ20 | O sistema deve permitir alterar do estado OPERATIONAL para o estado READY | Teste |
| SM_REQ21 | O sistema deve permitir alterar do estado READY para o estado STANDBY | Teste |
| SM_REQ22 | O sistema deve permitir alterar do estado READY para o estado EMERGENCY | Teste |
| SM_REQ23 | O sistema deve permitir alterar do estado FAIL para o estado REDUCED | Teste |
| SM_REQ24 | O sistema deve permitir alterar do estado REDUCED para o estado FAIL | Teste |
| SM_REQ25 | O sistema deve permitir alterar do estado FAIL para o estado EMERGENCY | Teste |
| SM_REQ26 | O sistema deve permitir alterar do estado FAIL para o estado STANDBY | Teste |
| SM_REQ27 | O sistema deve permitir alterar do estado MAINTENANCE para o estado EMERGENCY | Teste |
| SM_REQ28 | O sistema deve permitir alterar do estado OPERATIONAL para o estado EMERGENCY | Teste |
| SM_REQ29 | O sistema deve permitir alterar do estado REDUCED para o estado EMERGENCY | Teste |
| SM_REQ30 | O sistema deve permitir alterar do estado REDUCED para o estado STANDBY | Teste |
| SM_REQ31 | O sistema no estado OPERATIONAL deve iniciar no modo PANEL_CLOSE | Teste |
| SM_REQ32 | O sistema no estado REDUCED deve iniciar no modo PANEL_CLOSE | Teste |
| SM_REQ33 | O sistema deve permanecer no modo PANEL_CLOSE quando o painel se mantiver fechado. | Análise |
| SM_REQ34 | O sistema deve alterar do modo PANEL_CLOSE para EXTEND quando acionada a abertura do painel. | Teste |
| SM_REQ35 | O sistema no estado OPERATIONAL deve abrir o painel no modo EXTEND | Teste |
| SM_REQ36 | O sistema deve permanecer no modo PANEL_OPEN quando o painel se mantiver aberto. | Análise |
| SM_REQ37 | O sistema deve alterar do modo PANEL_OPEN para RETRACT quando acionado o fechamento do painel. | Teste |

Continua

Tabela 5.2 – Conclusão

| Requisito | Descrição | Método de Verificação |
|---|---|---|
| SM_REQ38 | O sistema no estado OPERATIONAL deve fechar o painel no modo RETRACT. | Teste |
| SM_REQ39 | O sistema no estado REDUCED deve fechar o painel no modo RETRACT. | Teste |
| SM_REQ40 | O sistema no estado REDUCED deve abrir o painel no modo EXTEND | Teste |
| SM_REQ41 | O sistema no estado OPERATIONAL deve ter opção para desligar em segurança, fechando o painel. | Teste |
| SM_REQ42 | O sistema no estado REDUCED deve ter opção para desligar em segurança, fechando o painel. | Teste |
| SM_REQ43 | No estado REDUCED a velocidade de abertura do painel é 50% da normal. | Análise |
| SM_REQ44 | O sistema deve monitorar, a todo ciclo, a potência gerada pelo painéis | Análise |
| SM_REQ45 | O sistema deve alterar seu estado para STANDBY ao detectar um estado inválido. | Teste |

Fonte: Produção do autor.

Tabela 5.3 – Estados.

| Estados | Descrição |
|---|---|
| STANDBY | Sistema aguardando início. |
| IBIT | Sistema realizando testes no sistema. |
| FAIL | Prepara sistema para trabalhar com falha. |
| READY | Sistema aguarda comando para entrar em operação. |
| REDUCED | Sistema opera com alguma falha. |
| MAINTENANCE | Sistema opera para manutenção. |
| OPERATIONAL | Sistema opera normalmente. |
| EMERGENCY | Sistema detectou falha. |

Fonte: Adaptado de Trivelato (2011).

Tabela 5.4 – Modos.

| Modos | Descrição |
|---|---|
| PANEL_CLOSE | Painel está fechado. |
| EXTEND | Sistema operando para abrir painel. |
| PANEL_OPEN | Painel está aberto. |
| RETRACT | Sistema operando para fechar painel. |
| SHUTDOWN | Sistema operando para desligar-se. |

Fonte: Adaptado de Trivelato (2011).

A Tabela 5.5, apresenta a associação entre os estados e modos que compõem a máquina de estados. Com exceção dos estados Operacional e Reduced, todos os demais estados apresentam modo de operação único.

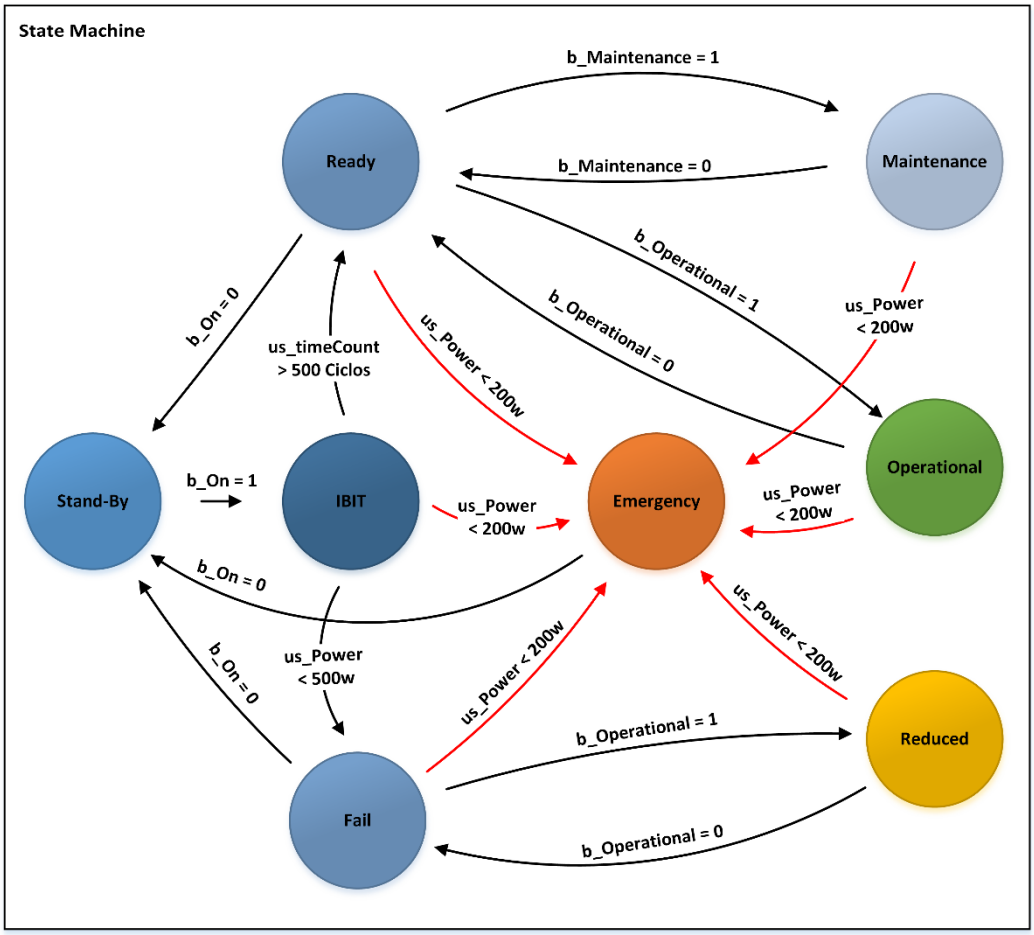
Tabela 5.5 – Associação de estados e modos.

| Estados | Modos |
|---|---|
| STANDBY | Apresenta modo único. |
| IBIT | Apresenta modo único. |
| FAIL | Apresenta modo único. |
| READY | Apresenta modo único. |
| MAINTENANCE | Apresenta modo único. |
| REDUCED | PANEL_CLOSE |
| | EXTEND |
| | PANEL_OPEN |
| | RETRACT |
| | SHUTDOWN |
| OPERATIONAL | PANEL_CLOSE |
| | EXTEND |
| | PANEL_OPEN |
| | RETRACT |
| | SHUTDOWN |
| EMERGENCY | Apresenta modo único. |

Fonte: Adaptado de Trivelato (2011).

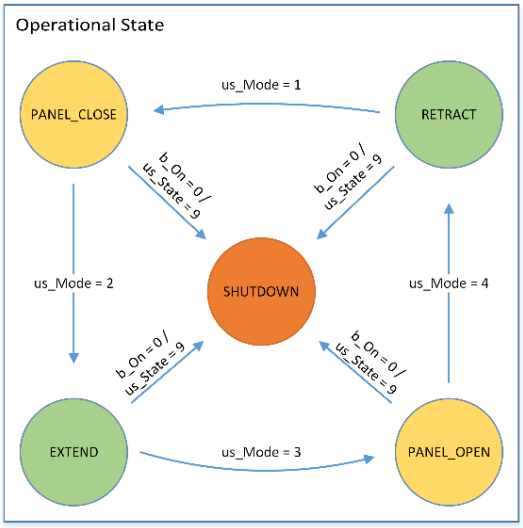
A Figura 5.2, ilustra a máquina de estados do componente de software e a Figura 5.3, ilustra os modos do estado OPERACIONAL. Enquanto a Tabela 5.6, apresenta a identificação das variáveis utilizadas no software, como base para melhor entendimento da descrição dos procedimentos de testes apresentados no ANEXO A – PROCEDIMENTOS DE TESTES.

Figura 5.2 – Diagrama de estados.




Fonte: Produção do autor.

Figura 5.3 – Diagrama de modos dos Estados Operacional e Reduced do software.




Fonte: Produção do autor.

Tabela 5.6 – Identificação das variáveis.

| Variável | Tipo | Descrição |
|---|---|---|
| b_Emergency | unsigned int | Indicador de sistema em emergência |
| b_Fail | unsigned int | Indicador de falha |
| b_Maintenance | unsigned int | Indicador de manutenção |
| b_On | unsigned int | Indicador de sistema ligado. |
| b_Operational | unsigned int | Indicador de operação |
| b_Panel | unsigned int | Indicador de estado do painel |
| ss_Position | signed int | Identificador da posição do painel |
| us_Mode | unsigned int | Identificador de modo de operação |
| us_Power | unsigned int | Indicador de potência do painel. |
| us_State | unsigned int | Identificador de estado de operação |
| us_timeCount | unsigned int | Contador de tempo |

Fonte: Produção do autor.

## 5.4. Atividades do processo de testes

Nesta seção apresentaremos os resultados encontrados na aplicação do processo proposto, conforme descrito detalhadamente na seção 4.3 - Visão Detalhada do Processo Proposto.

### 5.4.1. Estratégia de testes

O software utilizado neste experimento é devidamente representado através dos requisitos apresentados na Tabela 5.2, seus estados através da Tabela 5.3 e do diagrama representativo da máquina de estados apresentado na Figura 5.2, na Tabela 5.4 temos a identificação dos modos dos estados Operational e Reduced além do diagrama representativo da Figura 5.3. Na Tabela 5.5 temos associação de estados e modos, a identificação das variáveis apresentas na Tabela 5.6 e, por último, na Tabela 5.7, temos a associação entre os requisitos e os procedimentos de teste identificados para este experimento.

Conforme mencionado anteriormente, a estratégia definida para este experimento foi inicialmente a aplicação das análises estática e dinâmica no componente de software, precedendo a execução dos procedimentos de testes. Após a execução destes procedimentos verifica-se o atendimento e

cobertura dos requisitos de software, bem como, a cobertura da estrutura do código fonte obtida.

A verificação por análise, não está incluída no escopo deste trabalho ainda que alguns requisitos definidos na Tabela 5.2 estejam relacionados com este método de verificação.

### 5.4.2. Geração dos casos de testes

Os procedimentos de teste apresentados através da Tabela 5.7 foram gerados a partir dos requisitos da Tabela 5.2. A descrição detalhada apresenta os casos de teste, entradas, saídas esperadas, resultados e dados de cobertura de código, os quais podem ser encontrados no ANEXO A – PROCEDIMENTOS DE TESTES.

Tabela 5.7 – Requisitos e procedimentos associados.

| Requisito | Procedimento de Teste | Descrição |
|---|---|---|
| SM_REQ11 | SM_PT01 | Verifica a inicialização do sistema no estado STANDBY. |
| SM_REQ12 | SM_PT02 | Verifica a transição do estado STANBY para o estado IBIT. |
| SM_REQ14 | SM_PT03 | Verifica a transição do estado IBIT para o estado EMERGENCY ao detectar us_Power < 200 W. |
| SM_REQ15 | SM_PT04 | Verifica a transição do estado IBIT para o estado FAIL ao detectar us_Power abaixo de 500 W. |
| SM_REQ16 | SM_PT05 | Verifica a transição do estado IBIT para o estado READY ao detectar us_Power acima de 500 W. |
| SM_REQ17 | SM_PT06 | Verifica a transição do estado READY para o estado MAINTENANCE ao detectar b_Maintenance = 1. |
| SM_REQ18 | SM_PT07 | Verifica a transição do estado MAINTENANCE para o estado READY ao detectar b_Maintenance = 0. |
| SM_REQ19 | SM_PT08 | Verifica a transição do estado READY para o estado OPERATIONAL ao detectar b_Operational = 1. |



Continua

Tabela 5.7 - Continuação

| Requisito | Procedimento de Teste | Descrição |
|---|---|---|
| SM_REQ20 | SM_PT09 | Verifica a transição do estado OPERATIONAL para o estado READY ao detectar b_Operational = 0. |
| SM_REQ21 | SM_PT10 | Verifica a transição do estado READY para o estado STANDBY ao detectar b_On = 0. |
| SM_REQ22 | SM_PT11 | Verifica a transição do estado READY para o estado EMERGENCY ao detectar us_Power menor que 200 W. |
| SM_REQ23 | SM_PT12 | Verifica a transição do estado FAIL para o estado REDUCED ao detectar b_Operational = 1. |
| SM_REQ24 | SM_PT13 | Verifica a transição do estado REDUCED para o estado FAIL ao detectar b_Operational = 0. |
| SM_REQ25 | SM_PT14 | Verifica a transição do estado FAIL para o estado EMERGENCY ao detectar us_Power menor que 200 W. |
| SM_REQ26 | SM_PT15 | Verifica a transição do estado FAIL para o estado STANDBY ao detectar b_On = 0. |
| SM_REQ27 | SM_PT16 | Verifica a transição do estado MAINTENANCE para o estado EMERGENCY ao detectar us_Power menor que 200 W. |
| SM_REQ28 | SM_PT17 | Verifica a transição do estado OPERATIONAL para o estado EMERGENCY ao detectar us_Power menor que 200 W. |
| SM_REQ29 | SM_PT18 | Verifica a transição do estado REDUCED para o estado EMERGENCY ao detectar us_Power menor que 200 W. |
| SM_REQ30 | SM_PT19 | Verifica a transição do estado REDUCED para o estado STANDBY ao detectar b_On = 0. |
| SM_REQ31 | SM_PT20 | Verifica se o sistema inicia o estado OPERATIONAL no modo PANEL_CLOSE. |
| SM_REQ32 | SM_PT21 | Verifica se o sistema inicia o estado REDUCED no modo PANEL_CLOSE. |
| SM_REQ34 | SM_PT22 | Verifica no estado OPERATIONAL no modo PANEL_CLOSE para o modo EXTEND ao detectar b_Panel = 1. |
| SM_REQ35 | SM_PT23 | Verifica no estado OPERATIONAL a abertura do painel e transição do modo EXTEND para OPEN |

(Continua na próxima página.)



Continua

Tabela 5.7 - Conclusão

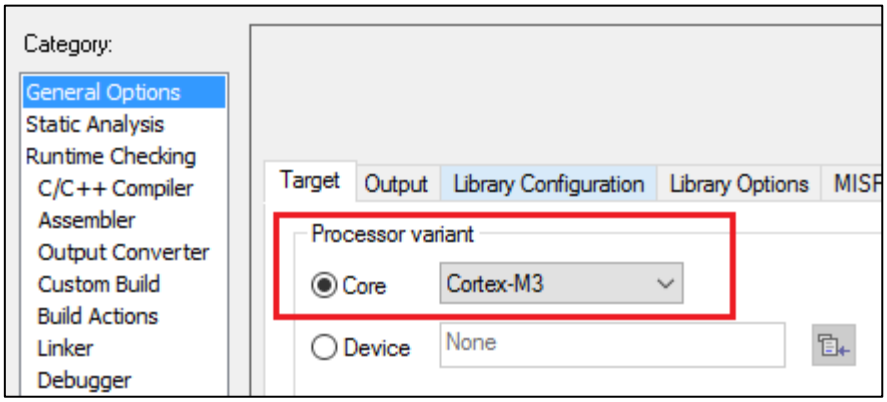
| Requisito | Procedimento de Teste | Descrição |
|---|---|---|
| SM_REQ37 | SM_PT24 | Verifica no estado OPERATIONAL a transição do modo OPEN para RETRACT ao detectar b_Panel = 0. |
| SM_REQ38 | SM_PT25 | Verifica no estado OPERATIONAL a transição do modo RETRACT para CLOSE |
| SM_REQ39 | SM_PT26 | Verifica no estado REDUCED a transição do modo RETRACT para PANEL_CLOSE |
| SM_REQ40 | SM_PT27 | Verifica no estado REDUCED a transição do modo PANEL_CLOSE para EXTEND |
| SM_REQ41 | SM_PT28 | Verifica a opção de SHUTDOWN em segurança no estado OPERATIONAL |
| SM_REQ42 | SM_PT29 | Verifica a opção de SHUTDOWN em segurança no estado REDUCED |
| SM_REQ45 | SM_PT30 | Verifica a transição para STANDBY ao detectar um estado inválido. |

Fonte: Produção do Autor.

### 5.4.3. Preparar ambiente para testes de integração hardware/software

A preparação do ambiente para os testes de integração hardware / software consiste em configurar a IDE IAR e a ferramenta TBrun. Isto inclui: configurar a IDE IAR para utilização de um hardware alvo (*target*), conforme ilustrado pela Figura 5.4, na qual se utiliza a simulação de um hardware Cortex-M3.

Figura 5.4 – Configuração do hardware.

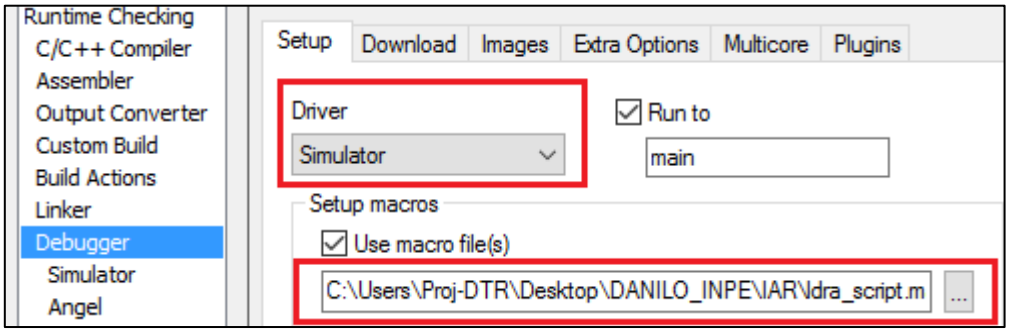



Fonte: IAR (2016).

Através da Figura 5.4, também é possível identificar que podemos selecionar a opção *Device* e definir um hardware físico configurado na ferramenta, neste caso os testes de integração de Hardware / Software seriam executados neste hardware.

A Figura 5.5, mostra a configuração do *Driver* do hardware, na qual se define o simulador e o arquivo que será usado pelo hardware simulado. O arquivo utilizado neste experimento foi cedido pela LDRA.

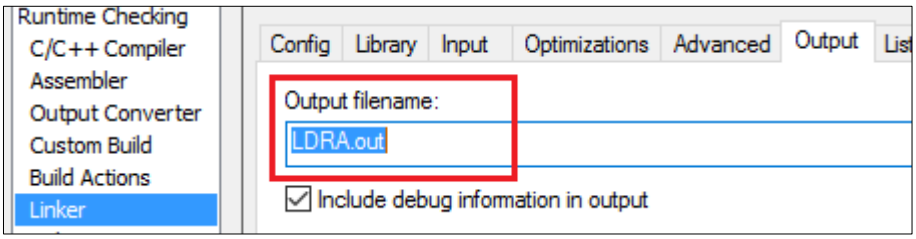
Figura 5.5 – Configuração do Driver do hardware.




Fonte: IAR (2016).

Após configurar o hardware, configura-se o arquivo que irá armazenar os dados retirados da execução do software no hardware (*Execution Data*), como mostrado na Figura 5.1. Como mostra a Figura 5.6, o arquivo de saída denominado “LDRA.out”, será lido pela ferramenta de testes para obter os dados sobre a cobertura da estrutura de código.

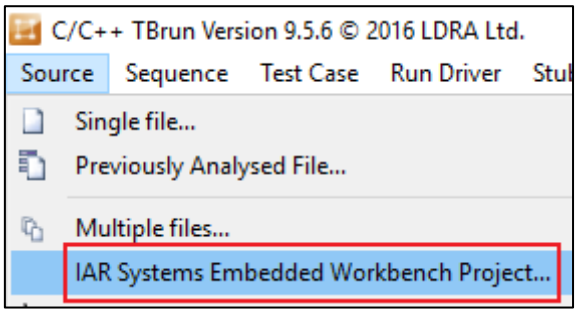
Figura 5.6 – Configuração do arquivo com dados de saída da IDE IAR.




Fonte: IAR (2016).

Após realizar as configurações na IDE IAR, pode-se carregar o projeto da IDE IAR no ambiente de testes, como mostra a Figura 5.7. Com isto, configura-se o local de descarregamento do código instrumentalizado e a localização do arquivo que contém os dados obtidos na execução dos testes, o qual gerado pela IDE IAR.

Figura 5.7 – Configuração do projeto na LDRA TBrun.



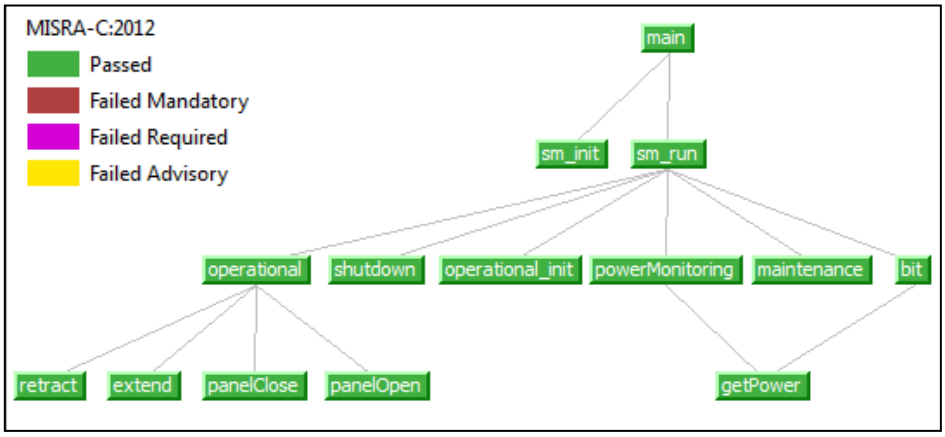

Fonte: LDRA TBrun (2016).

Para orientar as análises estática e dinâmica define-se na ferramenta LDRA TBrun, a norma DO-178C devido a seu rigor e disponibilidade e o padrão de codificação MISRA-C:2012, como um padrão moderno e muito usado no setor aeroespacial. Com o ambiente configurado, pode-se realizar as análises estática e dinâmica e a execução dos procedimentos de teste.

### 5.4.4. Executar teste de integração hardware / software

Como pré-requisito para a execução dos procedimentos de testes, temos a análise estática que verifica a estrutura do código fonte e o atendimento de algumas métricas. As figuras seguintes apresentam os resultados desta análise.

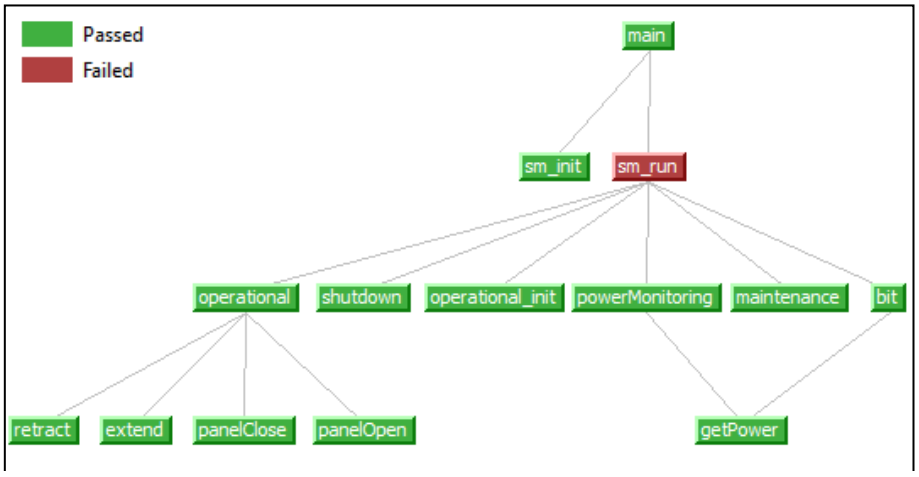
Figura 5.8 – Aderência ao padrão de código MISRA-C.




Fonte: LDRA TBrun (2016).

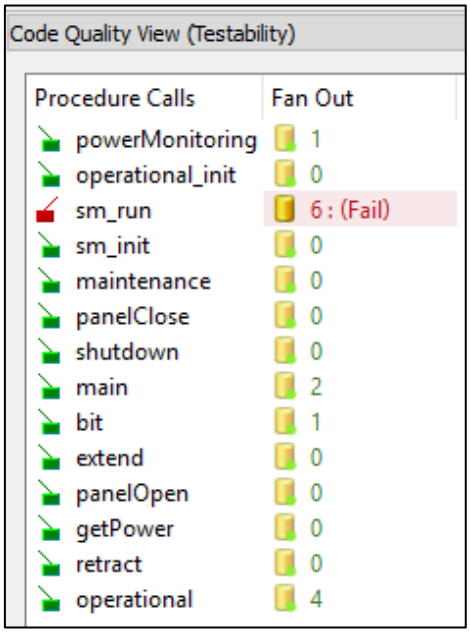
A Figura 5.8 mostra a aderência ao padrão MISRA-C:2012, na qual podemos identificar que todos os métodos do componente de software atendem aos requisitos deste padrão de codificação. A Figura 5.9 apresenta o resultado da análise estática para a métrica Testabilidade, na qual podemos identificar que o método “sm_run” não passou. Isto em decorrência da quantidade de métodos dependentes (*Fan-Out*) deste, conforme podemos identificar através dos detalhes apresentados na Figura 5.10.

Figura 5.9 – Testabilidade do código.




Fonte: LDRA TBrun (2016).

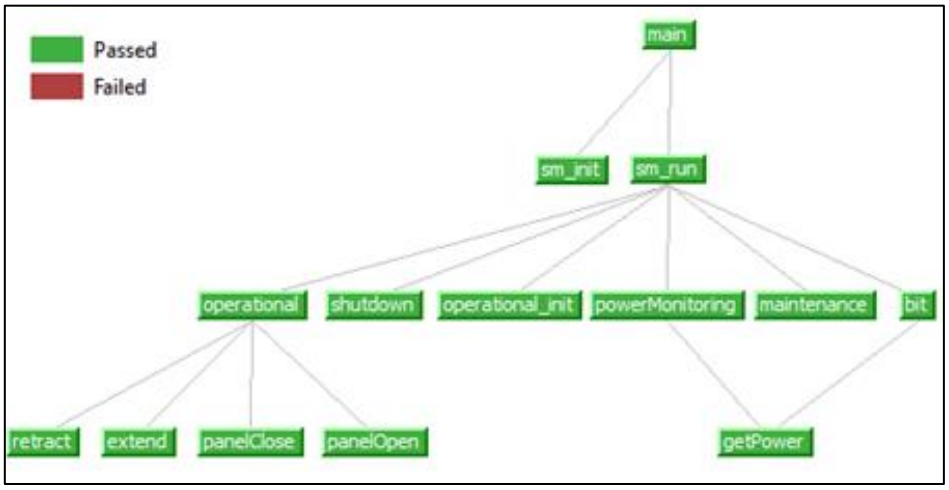
Figura 5.10 – Detalhes da métrica Testabilidade do código.




Fonte: LDRA TBrun (2016).

A Figura 5.11, apresenta o resultado da análise estática para a métrica de qualidade de código Clareza, indicando que o código fonte apresenta informações suficientes para seu entendimento.

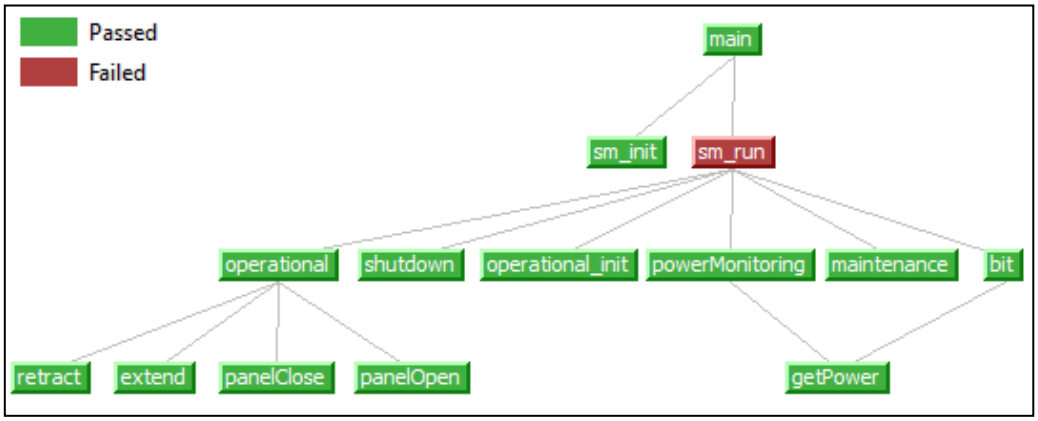
Figura 5.11 – Clareza do código.




Fonte: LDRA TBrun (2016).

A Figura 5.12, apresenta o resultado da análise estática para a métrica de qualidade Manutenabilidade do código fonte, na qual podemos identificar que o método “sm_run” novamente não passou.

Figura 5.12 – Manutenabilidade do código.




Fonte: LDRA TBrun (2016).

Através dos detalhes apresentados na Figura 5.13, identificamos que este método possui alta Complexidade Ciclomática e uma alta quantidade de Nós. Isto se deve a natureza da estrutura do código referente a máquina de estados. Melhorias para acomodar estas métricas poderiam impactar na segurança da máquina de estado e seu funcionamento para aplicações críticas e seguras.

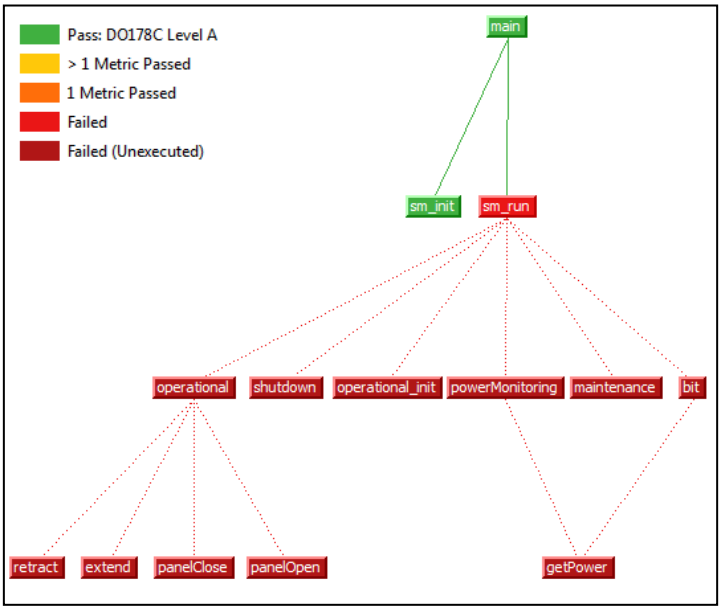
Figura 5.13 – Detalhes da manutenabilidade.

| Procedure Calls | Cyclomatic Complexity | Knots |
|---|---|---|
| powerMonitoring | 2 | 0 |
| operational_init | 1 | 0 |
| sm_run | 18 : (Fail) | 9 : (Fail) |
| sm_init | 1 | 0 |
| maintenance | 1 | 0 |
| panelClose | 2 | 0 |
| shutdown | 3 | 2 |
| main | 1 | 0 |
| bit | 4 | 2 |
| extend | 3 | 1 |
| panelOpen | 2 | 0 |
| getPower | 1 | 0 |
| retract | 3 | 1 |
| operational | 5 | 3 |

Fonte: LDRA TBrun (2016).

Quanto à análise dinâmica, a Figura 5.14 apresenta o resultado para a análise de cobertura do código inicial, segundo o tipo Cobertura de Declaração segundo as métricas da norma DO-178C.

Figura 5.14 – Cobertura de código inicial.




Fonte: LDRA TBrun (2016).

A Figura 5.15 apresenta os detalhes da análise de cobertura de código inicial, no qual identificamos que apenas dois métodos (sm_init e sm_main) atingiram a cobertura de 100% de suas estruturas.

Figura 5.15 – Detalhamento da cobertura do código inicial.

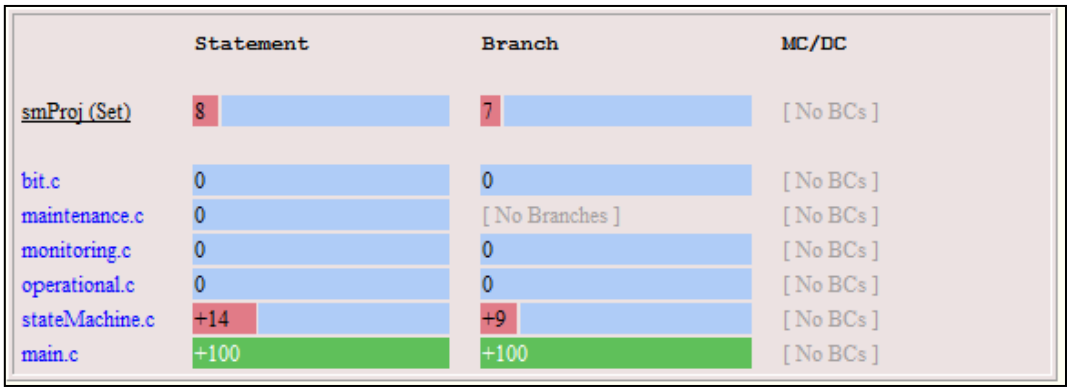
| Procedure Calls | Statement(100%) | Branch/Decision(100%) |
|---|---|---|
| powerMonitoring | 0 | 0 |
| operational_init | 0 | No Branches |
| sm_run | 23 | 21 |
| sm_init | 100 | No Branches |
| maintenance | 0 | No Branches |
| panelClose | 0 | 0 |
| shutdown | 0 | 0 |
| main | 100 | 100 |
| bit | 0 | 0 |
| extend | 0 | 0 |
| panelOpen | 0 | 0 |
| getPower | 0 | No Branches |
| retract | 0 | 0 |
| operational | 0 | 0 |

Fonte: LDRA TBrun (2016).

A Figura 5.16 apresenta a porcentagem de cobertura identificada pela análise dinâmica nos arquivos do componente de software. Conforme mostra o quadro,

a média da cobertura de declaração está em 8% e a média de cobertura de ramo em 7%.

Figura 5.16 – Cobertura após a análise dinâmica inicial.

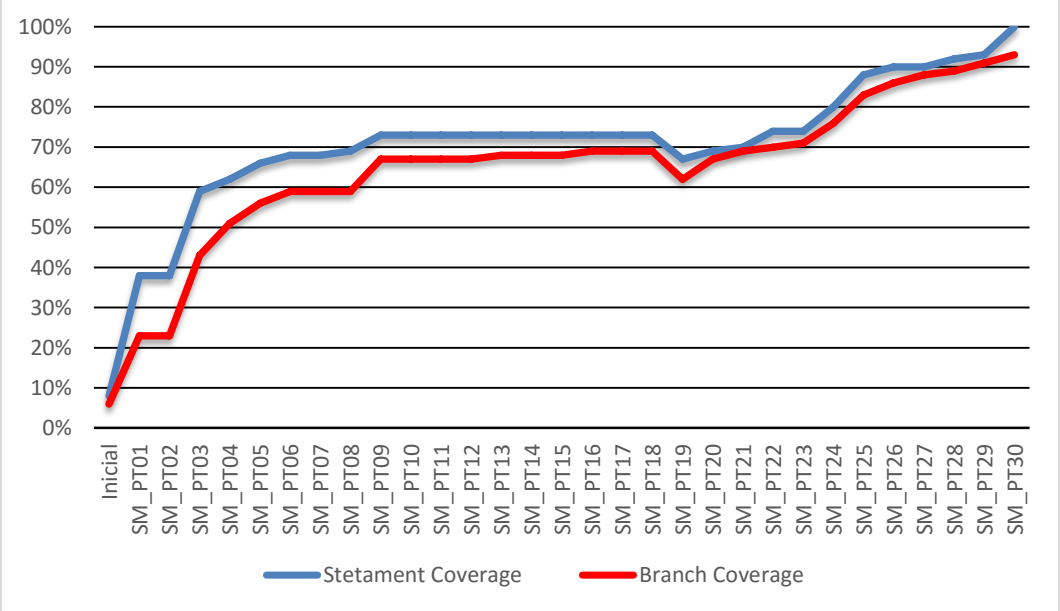


| | Statement | Branch | MC/DC |
|---|---|---|---|
| smProj (Set) | 8 | 7 | [ No BCs ] |
| bit.c | 0 | 0 | [ No BCs ] |
| maintenance.c | 0 | [ No Branches ] | [ No BCs ] |
| monitoring.c | 0 | 0 | [ No BCs ] |
| operational.c | 0 | 0 | [ No BCs ] |
| stateMachine.c | +14 | +9 | [ No BCs ] |
| main.c | +100 | +100 | [ No BCs ] |

Fonte: LDRA TBrun (2016).

Os procedimentos de testes foram executados conforme podem ser vistos em ANEXO A – PROCEDIMENTOS DE TESTES, o gráfico da Figura 5.17 mostra o avanço da cobertura de código para declaração e ramo, combinados com os resultados da análise dinâmica. O resultado e o cumprimento da cobertura de requisitos e da estrutura do código são avaliados na atividade seguinte.

Figura 5.17 – Avanço da cobertura estrutural.




Fonte: Produção do autor.

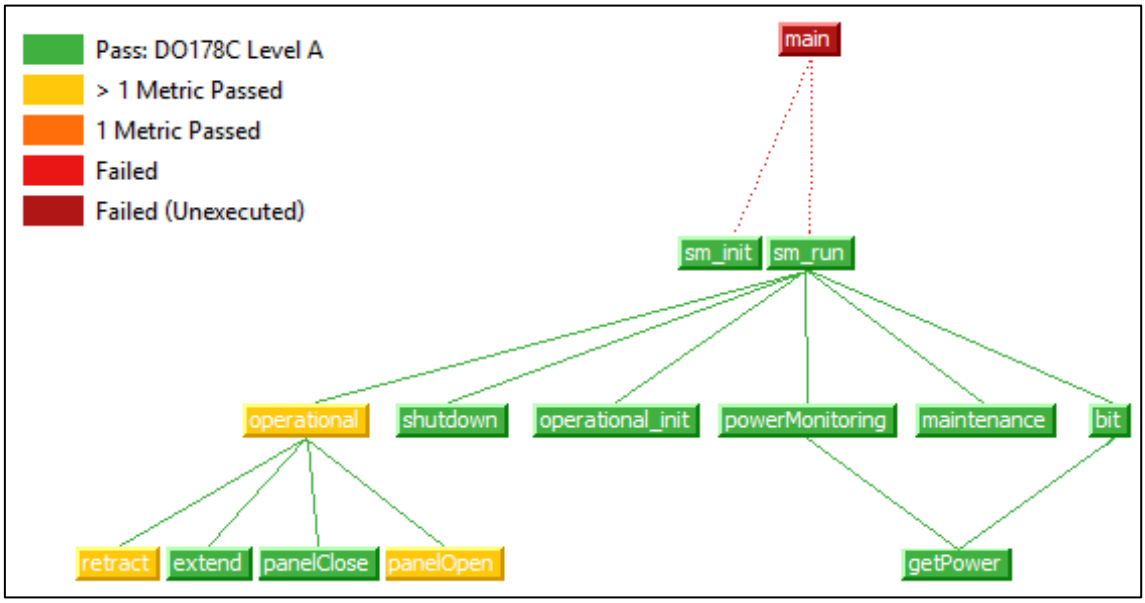
### 5.4.5. Análise de cobertura

Conforme ilustra o gráfico da Figura 5.17, todos os procedimentos de teste foram executados, os quais foram desenvolvidos com base nos requisitos. Entretanto, podemos verificar que com a execução de todos os procedimentos de teste alcançou-se um índice de 100% de cobertura para o tipo Cobertura de Declaração conforme as métricas da norma DO-178C, enquanto a cobertura do tipo Cobertura de Ramo atingiu 93%.

A Figura 5.18 mostra os procedimentos que não atingiram todas as métricas para cobertura de código, enquanto a Figura 5.19 detalha estes procedimentos e a cobertura alcançada para cobertura de Declaração e Ramo.

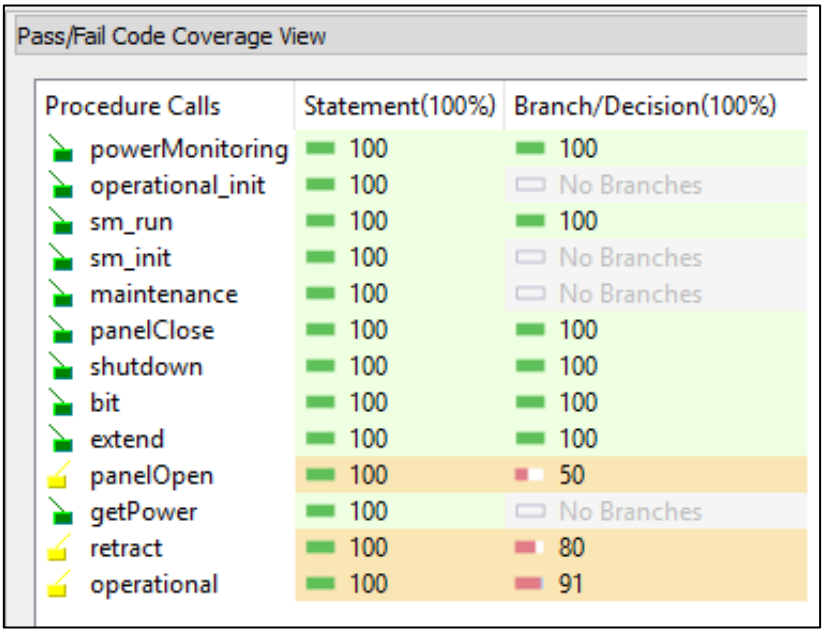
Conforme orienta o processo proposto, isto justifica uma avaliação dos requisitos e procedimentos de testes associados para verificar se há a necessidade de alterá-los ou criar novos procedimentos de teste com base nestes requisitos.

Figura 5.18 – Cobertura de código conforme a DO-178C.




Fonte: LDRA TBrun (2016).

Figura 5.19 – Detalhamento da cobertura dos procedimentos.




Fonte: LDRA TBrun (2016).

Deve-se desenvolver novos procedimentos de teste para complementar os procedimentos executados, os quais atingiram a cobertura de requisito, mas não foram suficientes para atingir 100% da estrutura do código, sendo este um requisito oriundo da norma DO-178C para o nível A.

Os procedimentos de teste que devem ser desenvolvidos, estão este no escopo dos testes de baixo nível de software, uma vez que verificam estruturas de repetição e estruturas de decisão. Com isto, observamos que não há a necessidade de desenvolver os testes de integração de software.

Está customização do processo proposto, deve-se ao fato de que o software utilizado possui uma estrutura de componentes simples e não apresenta complexidade e tamanho significativos. Com isto, os testes executados ao nível de integração de hardware / software foram suficientes para cobrir a estrutura do código, com exceção de pequenas partes que devem ser concluídas com a execução de procedimento de testes em baixo nível de software.

### 5.4.6. Geração dos casos de testes

A Tabela 5.8, apresenta os procedimentos de teste complementares, os quais foram desenvolvidos para atingir a cobertura estrutural do código fonte, especialmente, do tipo cobertura de ramo.

Tabela 5.8 – Procedimentos de teste complementares.

| Requisito | Caso de Teste Associado | Descrição |
|---|---|---|
| SM_REQ37 | PT_31 | Verifica no estado OPERATIONAL a transição do modo OPEN para RETRACT ao detectar b_Panel = 0, entretanto de modo complementar verifica a condição “FALSE” para o procedimento “panelOpen”. |
| SM_REQ38 | PT_32 | Verifica no estado OPERATIONAL a não transição do modo RETRACT para CLOSE, entretanto de modo complementar verifica a condição “FALSE“ para o procedimento “retract”. |

Fonte: Produção do autor.

### 5.4.7. Preparar Ambiente para teste de baixo nível

A preparação do ambiente de testes de baixo nível consiste em adicionar o novo procedimento de teste na ferramenta. Considerando as configurações anteriores, nenhuma nova configuração é necessária.

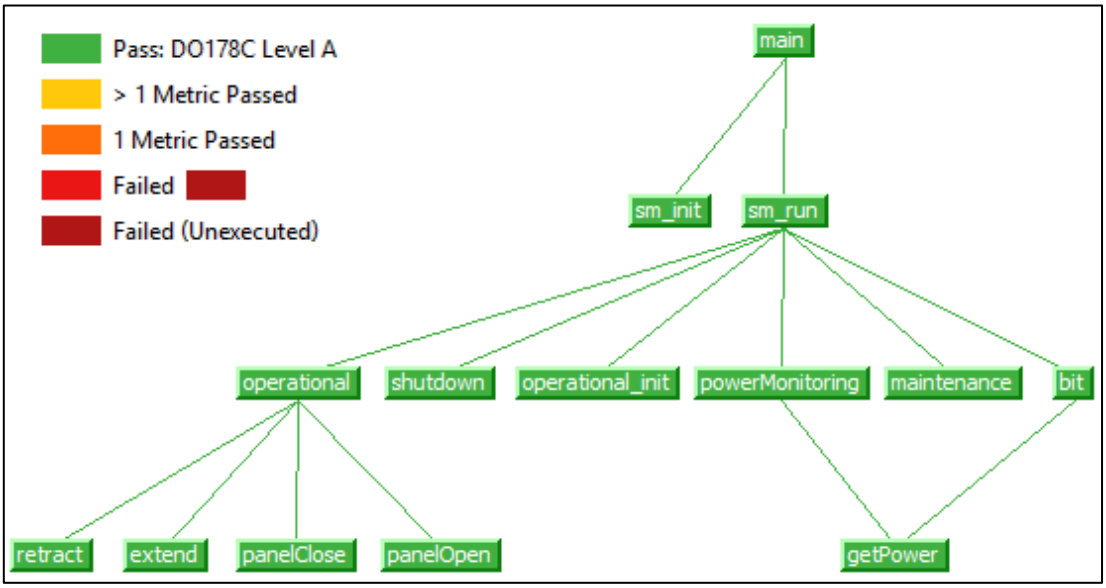
### 5.4.8. Executar testes de baixo nível de software

O procedimento de teste definido acima foi executado conforme pode ser visto no ANEXO A – PROCEDIMENTOS DE TESTES.

### 5.4.9. Análise de cobertura

Conforme apresentado pela Figura 5.20 e detalhado através da Figura 5.21 e Figura 5.22, podemos identificar que os procedimentos de teste desenvolvidos adicionalmente foram suficientes para atingir 100% de cobertura da estrutura do código, deste modo atingindo as métricas requeridas pela norma DO-178C para o nível A.

Figura 5.20 – Cobertura de procedimentos conforme a DO-178C.




Fonte: LDRA TBrun (2016).

Observamos também que a estrutura do tipo MC/DC não foi identificada no componente de software, logo não há índice de cobertura para este caso.

Figura 5.21 – Detalhamento da cobertura final dos procedimentos.

Pass/Fail Code Coverage View

| Procedure Calls | Statement(100%) | Branch/Decision(100%) |
|---|---|---|
| powerMonitoring | 100 | 100 |
| operational_init | 100 | No Branches |
| sm_run | 100 | 100 |
| sm_init | 100 | No Branches |
| maintenance | 100 | No Branches |
| panelClose | 100 | 100 |
| shutdown | 100 | 100 |
| main | 100 | 100 |
| bit | 100 | 100 |
| extend | 100 | 100 |
| panelOpen | 100 | 100 |
| getPower | 100 | No Branches |
| retract | 100 | 100 |
| operational | 100 | 100 |

Fonte: LDRA TBrun (2016).

Figura 5.22 – Cobertura final.

| Procedure | Statement | Branch | MC/DC |
|---|---|---|---|
| sMachine (Set) | 100 | 100 | [ No BCs ] |
| sm_init | 100 | [ No Branches ] | [ No BCs ] |
| sm_run | 100 | 100 | [ No BCs ] |
| main | +100 | +100 | [ No BCs ] |
| maintenance | +100 | [ No Branches ] | [ No BCs ] |
| getPower | 100 | [ No Branches ] | [ No BCs ] |
| powerMonitoring | 100 | 100 | [ No BCs ] |
| operational_init | 100 | [ No Branches ] | [ No BCs ] |
| operational | 100 | 100 | [ No BCs ] |
| extend | 100 | 100 | [ No BCs ] |
| retract | 100 | 100 | [ No BCs ] |
| panelClose | 100 | 100 | [ No BCs ] |
| panelOpen | 100 | 100 | [ No BCs ] |
| shutdown | 100 | 100 | [ No BCs ] |
| bit | +100 | +100 | [ No BCs ] |

Fonte: LDRA TBrun (2016).

## 5.5. Considerações Finais

Neste capítulo apresentamos a aplicação do processo de verificação por testes com o suporte de uma ferramenta especialista. As atividades necessárias foram desenvolvidas conforme o processo proposto, com o objetivo de avaliar como um todo a proposta apresentada com o uso de ferramentas de testes.

De um modo geral, os resultados demonstram ganhos em termos de eficiência e praticidade na aplicação dos testes, principalmente considerando a proposta de inversão da sequência de testes, uma vez que se elimina o esforço de repetir os testes nos níveis mais baixos, principalmente sem comprometer a qualidade da verificação. A utilização da ferramenta é fundamental na aplicação do processo, devido à possibilidade de reutilizar os dados dos testes entre os níveis, possibilitando um resultado combinado.

O desenvolvimento de um processo de verificação por testes com base nas normas aeroespaciais visa avaliar criticamente o software, isto devido ao rigor

encontrado nestas normas, ainda que isto traga benefícios sobre prazo e esforço empregado nestas atividades, embora este não seja o foco do processo proposto. Na busca por uma avaliação crítica e, principalmente, obter 100% da cobertura de código alguns pontos devem ser destacados, como o desenvolvimento de procedimentos testes e casos de testes mandatoriamente com base nos requisitos.

Quanto ao estudo de caso apresentado, possibilitou exercitar o processo proposto e identificar as demandas deste tipo de trabalho, no qual identificamos que a integração de ambientes geralmente possui restrições, as quais impactam diretamente os resultados finais, como por exemplo, as ferramentas precisam de acesso e conseguir operações de gravação e leitura da memória do hardware, o que é obtido em um ambiente altamente integrado.

Com este estudo também podemos identificar e realizar melhorias na arquitetura do software empregado no trabalho, uma vez que o software já havia sido utilizado e testado anteriormente, entretanto, não havia sido realizado um trabalho de verificação de sua estrutura quanto ao atendimento de métricas de qualidade de qualidade de software, atendimento a rígidos padrões de codificação e, sobretudo, a avaliação por uma ferramenta especialista e com tecnologia de ponta.

Significativas observações acerca da ferramenta serão apresentadas no próximo capítulo, como a conclusão, recomendações e sugestões para trabalhos futuros.

# 6 CONCLUSÕES, RECOMENDAÇÕES E SUGESTÕES PARA TRABALHOS FUTUROS

Este trabalho propos-se a estudar as normas de desenvolvimento de software das indústrias espacial e aeronáutica, respectivamente, representadas pelas normas ECSS-E-ST-40C e RTCA DO-178C, com o objetivo de apresentar uma proposta de um processo de verificação por testes baseado na comparação destas normas, sendo compatível com ambas as indústrias e com a realidade dos projetos desenvolvidos no INPE.

## 6.1. Contribuições do Trabalho

A partir da comparação das normas E-ST-40C e DO-178C, podemos mencionar que ambas possuem um equilibrado conjunto de processos. De um modo geral, a estrutura de processos apresentada por ambas as normas é semelhante, com distinções em pontos específicos, como é o caso do próprio processo de verificação por testes. Neste paralelo, a norma DO-178C se destaca ao apresentar maior clareza e rigor, oriundos da experiência aeronáutica.

O processo de verificação por testes proposto visa melhorar a qualidade dos produtos de software, reduzindo a quantidade de erros de código através da inversão da sequência de testes e a utilização de uma ferramenta especialista. Nesse contexto, buscando diminuir o esforço envolvido nas atividades, reduzindo custo e tempo, sem impactar a qualidade e a aderência às normas de desenvolvimento de software dos setores espacial e aeronáutico.

A aplicação do processo proposto mostrou que a inversão da sequência tradicional de testes apresenta significativos ganhos em termos de esforço, principalmente, na atividade de aplicação dos procedimentos de testes. A utilização de uma ferramenta de testes tem impacto direto no processo, pois, agiliza o desenvolvimento e aplicação dos procedimentos e casos de testes. Em realidade, só é obter as vantagens do processo proposto com o uso desse tipo de ferramenta. A maior contribuição da ferramenta é compartilhar os

resultados da aplicação de testes em um nível com os demais níveis (Hardware / Software, Software / Software e Unidade). Isto possibilita uma melhoria na gestão da cobertura de código que, juntamente com a inversão proposta, reduz o esforço aplicado na realização das atividades de teste.

Com isto, podemos destacar algumas das realizações deste trabalho:

- Desenvolvimento de um processo de testes que atende às normas aeroespaciais, com definição de atividades e uso de técnicas de testes amplamente utilizadas;
- Desenvolvimento de um estudo comparativo das normas RTCA DO-178C e ECSS-E-ST-40C;
- Implementação do processo com apoio do ambiente LDRA;
- Apresentação de uma sequência de testes inversa à tradicional, resultando em ganhos em termos de esforço e tempo.

## 6.2. Dificuldades Encontradas

Ao longo do desenvolvimento do trabalho muitas dificuldades foram identificadas. Algumas merecem destaque:

- Dificuldade na compreensão das normas aeroespaciais devido à falta de significativa experiência no setor;
- Ao longo do estudo foi necessário abandonar a discussão sobre alguns temas devido a extensão e considerável aumento do escopo do trabalho;
- A curva de aprendizagem para plena utilização das ferramentas representou uma dificuldade bastante incômoda ao final do trabalho. Mesmo com suporte especializado da empresa Konatus, a integração das ferramentas para o ambiente de teste não é trivial;

- Pequenos erros de código impossibilitam a execução dos testes iniciais, demostrando a importância de processos robustos de codificação e a utilização de um padrão de codificação.

## 6.3. Limitações e Aperfeiçoamento

A aplicação do processo proposto não exclui a utilização de revisões de código e induz a aplicação de testes unitários após a atividade de codificação de cada unidade no próprio ambiente. Na prática, isto possibilita um melhor aproveitamento da ferramenta, pois pequenos erros de código inviabilizam a aplicação dos testes e a integração das ferramentas.

Associadas a isto, melhorias no processo proposto quanto ao desenvolvimento de atividades de verificação que considerem os testes unitários após a codificação são consideráveis para a completude da verificação.

## 6.4. Considerações Finais e Trabalhos Futuros

Ao longo do desenvolvimento deste trabalho podemos verificar que as normas do conjunto ECSS, embora bem equilibradas encontram-se levemente defasadas ante, por exemplo, a norma aeronáutica DO-178C. Isto ao considerarmos como critérios a adoção de ferramentas de ponta, prática de novas abordagens e a incorporação de novas tecnologias de desenvolvimento.

Quanto ao estudo das normas, foi possível identificar que a norma DO-178C reduz a subjetividade e aborda técnicas modernas no desenvolvimento de software crítico e seguro sendo considerada, portanto, uma diretriz para desenvolvimento de software.

Este trabalho também mostrou o expressivo papel e contribuição das ferramentas semiautomáticas de teste no cumprimento e resultados do processo proposto. Embora o processo para estabelecer um ambiente de testes seja árduo, os resultados mostram que isto é superado pela contribuição e pela redução no esforço e quantidade de horas utilizadas pelas atividades de testes.

O uso de uma ferramenta especialista possibilitou gerir a execução dos procedimentos de testes e, principalmente, os resultados ao longo da aplicação para os três níveis de teste. Diante de critérios, como cobertura de requisitos e estrutura de código, a ferramenta demonstrou-se fundamental para o cumprimento do processo. Uma vez que através de recursos visuais da ferramenta é possível determinar a cada procedimento executado qual o índice de cobertura atingido, garantindo com total segurança o cumprimento de métricas para os mais altos níveis de segurança do software como, por exemplo, Nível A para DO-178C.

Diante da quantidade de temas encontrados ao longo do desenvolvimento deste trabalho e que, por motivo de tempo, não puderam ser abordados, identificamos algumas possibilidades de temas que complementariam este trabalho, dentre os quais destacamos os seguintes:

- Selecionar outros casos de teste representativos e mais complexos e aplicar todo o processo proposto a eles.
- Utilização da geração automática de casos de testes a partir de modelos conforme orienta a norma complementar DO-331.
- Utilização de outras ferramentas da suíte LDRA no desenvolvimento de atividades de verificação e rastreabilidade entre os requisitos e resultados dos testes.
- Utilização de práticas de integração e testes contínuos, fazendo uso de um ambiente altamente integrado com a suíte LDRA.
- Estudo compartivo de ambientes (ferramentas automáticas) de verificação de software.

## REFERÊNCIAS BIBLIOGRÁFICAS


ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS - ABNT (Brasil). (Org.). **ABNT NBR ISO/IEC 15288:2009**: engenharia de sistemas e software - processos de ciclo de vida de sistema. Rio de Janeiro, RJ, 2009.

ALBUQUERQUE, I. S.; PERONDI, L. F. Gerenciamento da configuração em projetos da área espacial. In: WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS, 1. (WETE), 2010, São José dos Campos. **Anais...** São José dos Campos: INPE, 2010. DVD. ISSN 2177-3114. Disponível em: <http://urlib.net/8JMKD3MGP7W/38UKP6H>. Acesso em: 23 jan. 2015.

ALBUQUERQUE, I. S.; PERONDI, L. F. Gestão da configuração e o ciclo de vida de um projeto na área espacial. In: WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS, 2. (WETE), São José dos Campos**. Anais...** São José dos Campos: INPE, 2011. DVD. ISSN 2236-2606. Disponível em:<http://urlib.net/J8LNKAN8RW/3BASJLL>. Acesso em: 18 jan. 2015.

ALONSO, J. D. D. **Guia para padronização do desenvolvimento de software crítico para aplicações espaciais**. São José dos Campos: ITA, 1998. 126 p. (CTA/ITA-IEC/TM-005/98). Disponível em: <http://www.bdita.bibl.ita.br/tesesdigitais/lista_resumo.php?num_tese=000418412>. Acesso em: 07 dez. 2014.

AMBROSIO, A. M. Teste de conformidade para software de sistemas espaciais. In: WORKSHOP DE TESES E DISSERTAÇÕES (WTD); LATIN AMERICAN SYMPOSIUM ON DEPENDABLE COMPUTING, 1., 2003, São Paulo. **Anais...** 2003. p. 7. (INPE-10240-PRE/5758). Disponível em:<http://urlib.net/sid.inpe.br/marciana/2004/04.07.08.41>. Acesso em: 12 jan. 2015.

AMBRÓSIO, A. M.; MATTIELLO-FRANCISCO, F.; CARDOSO, L. S.; SANTIAGO, V.; ARIAS, R.; VIJAYKUMAR, N. L.; LOUREIRO, G. **Experiências em projetos e uso de técnicas de verificação e validação de software em aplicações espaciais no INPE**. São José dos Campos: INPE, 2007. 45 p. (INPE-15182-NTC/374). Disponível em: <http://urlib.net/sid.inpe.br/mtc-m17@80/2008/05.08.12.33>. Acesso em: 11 jan. 2015.

AMBRÓSIO, A. M.; MATTIELLO-FRANCISCO, M. F.; MARTINS, E. An independent software verification and validation process for space applications.

In: CONFERENCE ON SPACE OPERATIONS 9., (SPACEOPS 2008), Hidelberg. **Proceedings...** 2008. p. 9. CD-ROM. (INPE-15303-PRE/10112).

BERNARDO, P. C.; KON, F. **A Importância dos testes automatizados -** controle ágil, rápido e confiável de qualidade. Engenharia de Software Magazine, v. 1, n. 3, p. 54-57, 2008.

BLACKBURN, M.; BUSSER, R.; NAUMAN, A.; MORGAN, T. **Using models for development and verification of high integrity systems**. < http://www.knowledgebytes.net/downloads/blackburn_incose_2004.pdf>

BLACKBURN, Mark et al. Model-based approach to security test automation. In: **Proceeding of Quality Week 2001**. T-VEC Technologies, Inc., 2001.

BLACKBURN, M. R.; BUSSER, R. D. T-VEC: a tool for developing critical system. In: INTERNATIONAL CONFERENCE ON COMPUTER ASSURANCE, 11., 1996, Sofia, Bulgaria. **Proceedings…** New York: ACM, 1996. p. 237-249.

BLACKBURN, M. R. Using models for test generation and analysis. In: DIGITAL AVIONICS SYSTEMS CONFERENCE, 17., 1998, Bellevue, WA. **Proceedings...** Bellevue, WA: The AIAA/IEEE/SAE, 1998. v. 1, p. C45/1-C45/8.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de softwares.** Rio de Janeiro, RJ: Elsevier, 2007. 394 ISBN 978-85-352-2634-8.

DEUTSCH, L. J. Resolving the cassini/huygens relay radio anomaly. In: AEROSPACE CONFERENCE, 2002, Pasadena. **Proceedings…** IEEE, 2002. Disponível em: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1035262. Acesso em: 21 jan. 2015.

PAULA JUNIOR, E. P. **Formulação e implementação de métodos e procedimentos para um processo de gestão de configuração aplicável a projetos espaciais e industriais**. 2013. 306 p. (sid.inpe.br/mtc-m19/2013/04.15.18.56-TDI). Dissertação (Mestrado em Engenharia e Gerenciamento de Sistemas Espaciais) - Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2013. Disponível em:<http://urlib.net/8JMKD3MGP7W/3DTFQNE>. Acesso em: 11 jul. 2015.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION - ECSS. **ECSS-E-ST-10C space engineering –** system engineering general requirements. Noordwijk, Netherlands, 2009.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION - ECSS. **ECSS-M-ST-40C space project management –** configuration and information management. Noordwijk, Netherlands, 2009.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION - ECSS**.** **ECSS-P-001 standardization policy**. Noordwijk, Netherlands, 2000.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION - ECSS. **ECSS-Q-ST-80C space product assurance –** software product assurance. Noordwijk, Netherlands, 2009.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION - ECSS. **ECSS-Q-ST-30C space product assurance:** dependability **–** software product assurance. Noordwijk, Netherlands, 2009.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION - ECSS. **ECSS-Q-ST-40C space product assurance:** safety **–** software product assurance. Noordwijk, Netherlands, 2009.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION- ECSS. **ECSS-E-ST-40C space engineering –** software. Noordwijk, Netherlands, 2009.

HECHT, M.; BUETTNER, D. Software testing in space programs. **CrossLink**, **Fall.** v.6, n.3, 2005. Disponível em: <http://www.aero.org/publications/crosslink/fall2005/06.html>. Acesso em: 29 jan. 2015.

JORGENSEN, P. C. **Software testing a craftsman's approach 2.** Boca Raton, FL: CRC, 2002. 359 ISBN 0-8493-0809-7

MAZZA, C. Standards: the foundation for Space I. T. In: WORKSHOP SPACE INFORMATION TECHNOLOGY IN THE 21TH CENTURY. European Space Operations, Darmstadt, Germany, Setembro 2000. Disponível em: <www.esoc.esa.de/pr/documents/workshops/it_2000/it_in_future/ESA_C_Mazza.ppt> Acesso em: 17 jun. 2015.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. John Wiley & Sons, 2011.

NIST. **Integration Definition for Functional Modeling (IDEF0)**. Gaithersburg, MD: National Institute for Standards and Technologies, 1993.

SOFTWARE QUALITY SYSTEM - SQS. **DO-178C/ED-12C. the new software standard for the avionic industry:** goal, changes, and challenges. Stollwerckstrasse, Cologne, Germany, 2012. Disponível em: <www.sqs.com/in/_download/DO-178C_ED-12C.pdf> Acesso em: 10 novembro 2014.

PEÑA, L. G.; SOUZA, M. L.O. Uma discussão sob o processo de verificação e validação de software crítico e seguro; definição de conceitos, limitações, normas, objetivos e técnicas aplicadas. In: WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS, 4. (WETE), 2013, São José dos Campos. **Anais...** São José dos Campos: INPE, 2013. On-line. Disponível em: <http://urlib.net/8JMKD3MGP5W34M/3F4BF68>. Acesso em: 05 dez. 2014.

REGINATO, J. P. M. **Uma proposta de aperfeiçoamento de um processo de gerenciamento de requisitos de sistema e de software e sua aplicação a sistemas espaciais e aeronáuticos embarcados**. 2012. 171 p. (sid.inpe.br/mtc-m19/2012/03.26.23.04-TDI). Dissertação (Mestrado em Engenharia e Gerenciamento de Sistemas Espaciais) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012. Disponível em:<http://urlib.net/8JMKD3MGP7W/3BJTJGH>. Acesso em: 02 fev. 2015.

REIS, M. F. S.; AMBROSIO, A. M.; FERREIRA, M. G. V. VVTeste: ambiente de geração e gerenciamento de testes e de defeitos como apoio aos processos de verificação e validação do MPS.Br. In: WORKSHOP ANUAL DE MELHORIA DE PROCESSO DO SOFTWARE BRASILEITO (MPS.BR), 8. – (WAMPS 2012), 2012, Itupeva, SP. 2012. p. 152-159.

RTCA INC. **DO-178C software considerations in airborne systems and equipment certification**, Washington, D.C., EUA, 2011.

RTCA INC. **DO-330 Software Tool Qualification and Considerations**, Washington, D.C., EUA, 2011.

RTCA INC. **DO-331 Model-Based Development and Verification Supplement to DO-178C and DO-278**, Washington, D.C., EUA, 2011.

RTCA INC. **DO-332 Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A**, Washington, D.C., EUA, 2011.

RTCA INC. **DO-333 Formal Methods Supplemnt to DO-178C and DO-278A**, Washington, D.C., EUA, 2011.

SAE INTERNATIONAL. **ARP-4754A guidelines for development of civil aircraft and systems**. Warrendale, PA, EUA, 2010.

SAE INTERNATIONAL**. ARP4761- Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment**. 1996.

SANTOS JUNIOR, D. **Implementação de processo de software para teste de equipamentos aeroespaciais**. 2007. 190 p. Dissertação (Mestrado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2007.

SOMMERVILLE, I. **Software engineering.** Boston, MA: Addison-Wesley, 2011. 773 ISBN 978-0-13-703515-1

TRIVELATO, G. C. **HCMS do rotor** – Especificação de Requisitos de Software (SRS). São José dos Campos, 2011. Relatório Técnico da Homine Informática Educação e Tecnologia. Não publicado.

TRIVELATO, G. C.; SOUZA, M. L. O. A discussion on the standard SAE-ARP-4754A and a proposal for using it in product certification and qualification of staff. In: SAE 2011 BRAZIL CONGRESS, March 2000, São Paulo, SP, Brazil. **Anais…** São Paulo, 2012. SAE2011-036-0387.

VECTOR SOFTWARE, INC**. Using VectorCAST/C++ with test driven development**. Vector software, 2014. Disponivel em: < https://www.vectorcast.com/resources/whitepapers/using-vectorcast-c-test-driven-development > Acesso em: 10 de novembro de 2015.

WILEY et al. **INCOSE systems engineering handbook**: a guide for system life cycle processes and activities. John Wiley & Sons, 2015.

YASSUDA, I. S. **Artefatos de categorização de projetos espaciais e seleção de metodologias de gestão**. 2013. 146 p. (sid.inpe.br/mtc-m19/2013/08.13.04.48-TDI). Tese (Doutorado em Engenharia e Gerenciamento de Sistemas Espaciais) - Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2013. Disponível em: <http://urlib.net/8JMKD3MGP7W/3EKT272>. Acesso em: 18 nov. 2014.

YASSUDA, I. S. **Ciclo de vida de projetos na área espacial**. São José dos Campos: INPE, 2010. 32 p. (sid.inpe.br/mtc-m19@80/2010/03.02.19.04-PUD). Disponível em: <http://urlib.net/8JMKD3MGP7W/3746NA8>. Acesso em: 07 dez. 2014.

YASSUDA, I. S.; PERONDI, L. F. Estudo comparativo entre a gestão de projetos no setor espacial conforme o padrão ECSS e projetos realizados pelo INPE. In: WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS, 1. (WETE), 2010, São José dos Campos. **Anais...** São José dos Campos: INPE, 2010. v. IWETE2010-1021. CD-ROM. ISSN 2177-3114. Disponível em: <http://urlib.net/8JMKD3MGP7W/399KHUF>. Acesso em: 27 nov. 2014.

## ANEXO A – PROCEDIMENTOS DE TESTES

Tabela A.1 – Procedimento de teste SM_PT01.

| **Descrição** | Verifica a inicialização do sistema no estado STANDBY. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT01_01 | sm_run | us_State | 1 | 1 | 1 | OK |
| Resultado | Statement Coverage – 21%<br>Branch Coverage – 19% | | | | | |

Fonte: Produção do autor.

Tabela A.2 – Procedimento de teste SM_PT02.

| **Descrição** | Verifica a transição do estado STANBY para o estado IBIT. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT02_01 | sm_run | us_State | 1 | 2 | 2 | OK |
| | | b_On | 1 | 1 | 1 | OK |
| Resultado | Statement Coverage – 21%<br>Branch Coverage – 19% | | | | | |

Fonte: Produção do autor.

Tabela A.3 – Procedimento de teste SM_PT03.

| **Descrição** | Verifica a transição do estado IBIT para o estado EMERGENCY ao detectar us_Power < 200 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT03_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT03_02 | sm_run | us_State | 2 | 9 | 9 | OK |
| | | b_On | 1 | 1 | 1 | |
| PT03_03 | sm_run | us_State | 9 | 1 | 1 | OK |
| | | b_On | 1 | 0 | 0 | |
| | | b_Emergency | 0 | 1 | 1 | |
| **Resultado** | Statement Coverage – 32%<br>Branch Coverage – 25% | | | | | |

Fonte: Produção do autor.

Tabela A.4 – Procedimento de teste SM_PT04.

| **Descrição** | Verifica a transição do estado IBIT para o estado FAIL ao detectar us_Power abaixo de 500 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT04_01 | getPower | us_Power | 300 | 300 | 300 | OK |
| PT04_02 | sm_run | us_State | 2 | 3 | 3 | OK |
| | | b_On | 1 | 1 | 1 | |
| PT04_03 | sm_run | us_State | 3 | 3 | 3 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Fail | 0 | 1 | 1 | |
| **Resultado** | Statement Coverage – 39%<br>Branch Coverage – 34% | | | | | |

Fonte: Produção do autor.

Tabela A.5 – Procedimento de teste SM_PT05.

| **Descrição** | Verifica a transição do estado IBIT para o estado READY ao detectar us_Power acima de 500 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT05_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT05_02 | sm_run | us_State | 2 | 4 | 4 | OK |
| | | us_timeCount | 50 | 0 | 0 | |
| PT05_03 | sm_run | us_State | 4 | 4 | 4 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 42%<br>Branch Coverage – 37% | | | | | |

Fonte: Produção do autor.

Tabela A.6 – Procedimento de teste SM_PT06.

| **Descrição** | Verifica a transição do estado READY para o estado MAINTENANCE ao detectar b_Maintenance = 1. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT06_01 | sm_run | us_State | 4 | 6 | 6 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Maintenance | 1 | 1 | 1 | |
| **Resultado** | Statement Coverage – 48%<br>Branch Coverage – 36% | | | | | |

Fonte: Produção do autor.

Tabela A.7 – Procedimento de teste SM_PT07.

| **Descrição** | Verifica a transição do estado MAINTENANCE para o estado READY ao detectar b_Maintenance = 0. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT07_01 | sm_run | us_State | 6 | 4 | 4 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Maintenance | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 51%<br>Branch Coverage – 49% | | | | | |

Fonte: Produção do autor.

Tabela A.8 – Procedimento de teste SM_PT08.

| **Descrição** | Verifica a transição do estado READY para o estado OPERATIONAL ao detectar b_Operational = 1. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT08_01 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| **Resultado** | Statement Coverage – 54%<br>Branch Coverage – 52% | | | | | |

Fonte: Produção do autor.

Tabela A.9 – Procedimento de teste SM_PT09.

| **Descrição** | Verifica a transição do estado OPERATIONAL para o estado READY ao detectar b_Operational = 0. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT09_01 | sm_run | us_State | 7 | 4 | 4 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 59%<br>Branch Coverage – 58% | | | | | |

Fonte: Produção do autor.

Tabela A.10 – Procedimento de teste SM_PT10.

| **Descrição** | Verifica a transição do estado READY para o estado STANDBY ao detectar b_On = 0. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT10_01 | sm_run | us_State | 4 | 1 | 1 | OK |
| | | b_On | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 60%<br>Branch Coverage – 59% | | | | | |

Fonte: Produção do autor.

Tabela A.11 – Procedimento de teste SM_PT11.

| **Descrição** | Verifica a transição do estado READY para o estado EMERGENCY ao detectar us_Power menor que 200 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT11_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT11_02 | sm_run | us_State | 4 | 9 | 9 | OK |
| PT11_03 | sm_run | b_Emergency | 0 | 1 | 1 | OK |
| | | b_On | 1 | 0 | 0 | |
| | | b_state | 9 | 1 | 1 | |
| **Resultado** | Statement Coverage – 60%<br>Branch Coverage – 59% | | | | | |

Fonte: Produção do autor.

Tabela A.12 – Procedimento de teste SM_PT12.

| **Descrição** | Verifica a transição do estado FAIL para o estado REDUCED ao detectar b_Operational = 1. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT12_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT12_02 | sm_run | us_State | 3 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Fail | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| **Resultado** | Statement Coverage – 61%<br>Branch Coverage – 61% | | | | | |

Fonte: Produção do autor.

Tabela A.13 – Procedimento de teste SM_PT13.

| **Descrição** | Verifica a transição do estado REDUCED para o estado FAIL ao detectar b_Operational = 0. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT13_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT13_02 | sm_run | us_State | 7 | 3 | 3 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Fail | 1 | 1 | 1 | |
| | | b_Operational | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 62%<br>Branch Coverage – 63% | | | | | |

Fonte: Produção do autor.

Tabela A.14 – Procedimento de teste SM_PT14.

| **Descrição** | Verifica a transição do estado FAIL para o estado EMERGENCY ao detectar us_Power menor que 200 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT14_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT14_02 | sm_run | us_State | 3 | 9 | 9 | OK |
| PT14_03 | sm_run | us_State | 9 | 1 | 1 | OK |
| | | b_On | 1 | 0 | 0 | |
| | | b_Emergency | 0 | 1 | 1 | |
| **Resultado** | Statement Coverage – 62%<br>Branch Coverage – 63% | | | | | |

Fonte: Produção do autor.

Tabela A.15 – Procedimento de teste SM_PT15.

| **Descrição** | Verifica a transição do estado FAIL para o estado STANDBY ao detectar b_On = 0. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT15_01 | getPower | us_Power | 500 | 500 | 500 | OK |
| PT15_02 | sm_run | us_State | 3 | 1 | 1 | OK |
| | | b_On | 0 | 0 | 0 | |
| | | b_Fail | 1 | 0 | 0 | |
| **Resultado** | Statement Coverage – 63%<br>Branch Coverage – 64% | | | | | |

Fonte: Produção do autor.

Tabela A.16 – Procedimento de teste SM_PT16.

| **Descrição** | Verifica a transição do estado MAINTENANCE para o estado EMERGENCY ao detectar us_Power menor que 200 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT16_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT16_02 | sm_run | us_State | 6 | 9 | 9 | OK |
| PT16_03 | sm_run | us_State | 9 | 1 | 1 | OK |
| | | b_On | 1 | 0 | 0 | |
| | | b_Emergency | 0 | 1 | 1 | |
| **Resultado** | Statement Coverage – 63%<br>Branch Coverage – 64% | | | | | |

Fonte: Produção do autor.

Tabela A.17 – Procedimento de teste SM_PT17.

| **Descrição** | Verifica a transição do estado OPERATIONAL para o estado EMERGENCY ao detectar us_Power menor que 200 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT17_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT17_02 | sm_run | us_State | 7 | 9 | 9 | OK |
| PT17_03 | sm_run | b_Emergency | 0 | 1 | 1 | OK |
| | | b_On | 1 | 0 | 0 | |
| | | us_State | 9 | 1 | 1 | |
| **Resultado** | Statement Coverage – 63%<br>Branch Coverage – 64% | | | | | |

Fonte: Produção do autor.

Tabela A.18 – Procedimento de teste SM_PT18.

| **Descrição** | Verifica a transição do estado REDUCED para o estado EMERGENCY ao detectar us_Power menor que 200 W. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT18_01 | getPower | us_Power | 100 | 100 | 100 | OK |
| PT18_02 | sm_run | us_State | 7 | 9 | 9 | OK |
| PT18_03 | sm_run | us_State | 9 | 1 | 1 | OK |
| | | b_On | 1 | 0 | 0 | |
| | | b_Fail | 1 | 0 | 0 | |
| | | b_Emergency | 0 | 1 | 1 | |
| **Resultado** | Statement Coverage – 63%<br>Branch Coverage – 64% | | | | | |

Fonte: Produção do autor.

Tabela A.19 – Procedimento de teste SM_PT19.

| **Descrição** | Verifica a transição do estado REDUCED para o estado STANDBY ao detectar b_On = 0. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT19_01 | getPower | us_Power | 500 | 500 | 500 | OK |
| PT19_02 | sm_run | us_State | 7 | 1 | 1 | OK |
| | | b_On | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 68%<br>Branch Coverage – 68% | | | | | |

Fonte: Produção do autor.

Tabela A.20 – Procedimento de teste SM_PT20.

| **Descrição** | Verifica se o sistema inicia o estado OPERATIONAL no modo PANEL_CLOSE. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT20_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT20_02 | operational | b_Panel | 0 | 0 | 0 | OK |
| | | ss_Position | 0 | 0 | 0 | |
| | | us_Mode | 1 | 1 | 1 | |
| PT20_03 | sm_run | us_State | 7 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT20_04 | operational | b_Fail | 0 | 0 | 0 | OK |
| | | us_Mode | 1 | 1 | 1 | |
| PT20_05 | operational | b_Panel | 0 | 0 | 0 | OK |
| | | us_Mode | 1 | 1 | 1 | |
| **Resultado** | Statement Coverage – 69%<br>Branch Coverage – 70% | | | | | |

Fonte: Produção do autor.

Tabela A.21 – Procedimento de teste SM_PT21.

| **Descrição** | Verifica se o sistema inicia o estado REDUCED no modo PANEL_CLOSE. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT21_01 | getPower | us_Power | 400 | 400 | 400 | OK |
| PT21_02 | operational | b_Panel | 0 | 0 | 0 | OK |
| | | ss_Position | 0 | 0 | 0 | |
| | | us_Mode | 1 | 1 | 1 | |
| PT21_03 | sm_run | us_State | 7 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT21_04 | operational | b_Fail | 1 | 1 | 1 | OK |
| | | us_Mode | 1 | 1 | 1 | |
| PT21_05 | operational | b_Panel | 0 | 0 | 0 | OK |
| | | us_Mode | 1 | 1 | 1 | |
| **Resultado** | Statement Coverage – 70%<br>Branch Coverage – 70% | | | | | |

Fonte: Produção do autor.

Tabela A.22 – Procedimento de teste SM_PT22.

| **Descrição** | Verifica no estado OPERATIONAL a transição do modo PANEL_CLOSE para o modo EXTEND ao detectar b_Panel = 1. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT22_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT22_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT22_03 | operational | b_Fail | 0 | 0 | 0 | OK |
| | | b_Panel | 0 | 0 | 0 | |
| | | us_Mode | 1 | 1 | 1 | |
| PT22_04 | operational | b_Panel | 1 | 1 | 1 | OK |
| | | us_Mode | 1 | 2 | 2 | |
| **Resultado** | Statement Coverage – 75%<br>Branch Coverage – 71% | | | | | |

Fonte: Produção do autor.

Tabela A.23 – Procedimento de teste SM_PT23.

| **Descrição** | Verifica no estado OPERATIONAL a abertura do painel e transição do modo EXTEND para OPEN. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT23_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT23_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| | | b_Panel | 1 | 1 | 1 | |
| | | us_Mode | 2 | 3 | 3 | |
| | | ss_Position | 19 | 21 | 21 | |
| **Resultado** | Statement Coverage – 86%<br>Branch Coverage – 83% | | | | | |

Fonte: Produção do autor.

Tabela A.24 – Procedimento de teste SM_PT24.

| **Descrição** | Verifica no estado OPERATIONAL a transição do modo OPEN para RETRACT ao detectar b_Panel = 0. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT24_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT24_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT24_03 | operational | us_Mode | 3 | 4 | 4 | OK |
| | | b_panel | 0 | 0 | 0 | |
| | | b_Fail | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 90%<br>Branch Coverage – 85% | | | | | |

Fonte: Produção do autor.

Tabela A.25 – Procedimento de teste SM_PT25.

| **Descrição** | Verifica no estado OPERATIONAL a transição do modo RETRACT para CLOSE. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT25_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT25_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT25_03 | operational | us_Mode | 4 | 1 | 1 | OK |
| | | ss_Position | 1 | -1 | -1 | |
| | | b_Fail | 0 | 0 | 0 | |
| **Resultado** | Statement Coverage – 90%<br>Branch Coverage – 85% | | | | | |

Fonte: Produção do autor.

Tabela A.26 – Procedimento de teste SM_PT26.

| **Descrição** | Verifica no estado REDUCED a transição do modo RETRACT para PANEL_CLOSE. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT26_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT26_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT26_03 | operational | ss_Position | 0 | -1 | -1 | OK |
| | | b_Fail | 1 | 1 | 1 | |
| PT26_04 | operational | b_Panel | 1 | 2 | 2 | OK |
| **Resultado** | Statement Coverage – 92%<br>Branch Coverage – 88% | | | | | |

Fonte: Produção do autor.

Tabela A.27 – Procedimento de teste SM_PT27.

| **Descrição** | Verifica no estado REDUCED a transição do modo PANEL_CLOSE para EXTEND. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT27_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT27_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT27_03 | operational | us_Mode | 1 | 1 | 1 | OK |
| | | b_Panel | 0 | 0 | 0 | |
| | | b_Fail | 1 | 1 | 1 | |
| PT27_03 | operational | b_Panel | 1 | 2 | 2 | OK |
| **Resultado** | Statement Coverage – 93%<br>Branch Coverage – 90% | | | | | |

Fonte: Produção do autor.

Tabela A.28 – Procedimento de teste SM_PT28.

| **Descrição** | Verifica a opção de SHUTDOWN em segurança no estado OPERATIONAL. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT28_01 | getPower | us_Power | 1000 | 1000 | 1000 | OK |
| PT28_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT28_03 | operational | ss_Position | 0 | 0 | 0 | OK |
| | | us_Mode | 1 | 1 | 1 | |
| | | b_Panel | 0 | 0 | 0 | |
| PT28_04 | sm_run | us_State | 7 | 4 | 4 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 0 | 0 | 0 | |
| PT28_05 | shutdown | ss_Position | 20 | 0 | 0 | OK |
| **Resultado** | Statement Coverage – 97%<br>Branch Coverage – 93% | | | | | |

Fonte: Produção do autor.

Tabela A.29 – Procedimento de teste SM_PT29.

| **Descrição** | Verifica a opção de SHUTDOWN em segurança no estado REDUCED. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT29_01 | getPower | us_Power | 400 | 400 | 400 | OK |
| PT29_02 | sm_run | us_State | 4 | 7 | 7 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 1 | 1 | 1 | |
| PT29_03 | operational | b_Fail | 1 | 1 | 1 | OK |
| | | ss_Position | 0 | 0 | 0 | |
| | | us_Mode | 1 | 1 | 1 | |
| | | b_Panel | 0 | 0 | 0 | |
| PT29_04 | sm_run | us_State | 7 | 4 | 4 | OK |
| | | b_On | 1 | 1 | 1 | |
| | | b_Operational | 0 | 0 | 0 | |
| PT29_05 | shutdown | b_Fail | 1 | 1 | 1 | OK |
| | | ss_Position | 20 | 0 | 0 | |
| **Resultado** | Statement Coverage – 99%<br>Branch Coverage – 95% | | | | | |

Fonte: Produção do autor.

Tabela A.30 – Procedimento de teste SM_PT30.

| **Descrição** | Verifica a robustez do sistema para variável us_State utilizando valor fora do range. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| SM_PT30_01 | sm_run | us_State | 10 | 1 | 1 | OK |
| **Resultado** | Statement Coverage – 100%<br>Branch Coverage – 93% | | | | | |

Fonte: Produção do autor.

Tabela A.31 – Procedimento de teste SM_PT31.

| **Descrição** | Verifica no estado OPERATIONAL a transição do modo OPEN para RETRACT ao detectar b_Panel = 0, entretanto de modo complementar verifica a condição “FALSE” para o procedimento “panelOpen”. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT31_01 | getPower | us_State | 1000 | 1000 | 1000 | OK |
| PT31_02 | sm_run | b_On | 1 | 1 | 1 | OK |
| | | b_Operational | 1 | 1 | 1 | |
| | | us_State | 4 | 7 | 7 | |
| PT31_03 | panelOpen | b_Fail | 0 | 0 | 0 | OK |
| | | b_Panel | 1 | 1 | 1 | |
| | | us_Mode | 3 | 3 | 3 | |
| **Resultado** | Statement Coverage – 100%<br>Branch Coverage – 97% | | | | | |

Fonte: Produção do autor.

Tabela A.32 – Procedimento de teste SM_PT32.

| **Descrição** | Verifica no estado OPERATIONAL a não transição do modo RETRACT para CLOSE, entretanto de modo complementar verifica a condição “FALSE“ para o procedimento “retract”. | | | | | |
|---|---|---|---|---|---|---|
| **Caso de Teste** | **Procedimento** | **Variável** | **Valor Entrada** | **Valor Saída** | **Valor Atual** | **Status** |
| PT32_01 | getPower | us_State | 1000 | 1000 | 1000 | OK |
| PT32_02 | sm_run | b_On | 1 | 1 | 1 | OK |
| | | b_Operational | 1 | 1 | 1 | |
| | | us_State | 4 | 7 | 7 | |
| PT33_03 | retract | b_Fail | 0 | 0 | 0 | OK |
| | | ss_Position | 1 | -1 | -1 | |
| | | us_Mode | 4 | 4 | 4 | |
| **Resultado** | Statement Coverage – 100%<br>Branch Coverage – 100% | | | | | |

Fonte: Produção do autor.

## ANEXO B – CÓDIGO FONTE

```
/*****************************************************************
 * File: main.c
 * Author: @DaniloGraca
 *****************************************************************/

#include "stateMachine.h"

/*
 * Main procedure
 * Return: Void
 */
int main(void) {

    while (1) {
        sm_init(); /* Initialize all the variables */
        sm_run(); /* Execute the state machine */
    }
}

/*****************************************************************
 * File: stateMachine.c
 * Author: @DaniloGraca
*****************************************************************/

#include "bit.h"
#include "maintenance.h"
#include "operational.h"
#include "monitoring.h"

static UINT_16 b_On; /* System Control Turn ON/OFF */
static UINT_16 b_Fail; /* Control fail active */
static UINT_16 b_Maintenance; /* Control system maintenance
active */
static UINT_16 b_Emergency; /* Control system emergency active
*/
static UINT_16 b_Operational; /* Control operational active */
static UINT_16 us_State; /* Storage currently state */

/*
 * Variables Initialization
 * Called by: Main.c
 * Return: void
 */
void sm_init(void) {
    us_State = 1U; /* System init on State 1 */
    b_On = 0U; /* All controls start deactivated */
    b_Fail = 0U; /* All controls start deactivated */
    b_Operational = 0U; /* All controls start deactivated */
    b_Maintenance = 0U; /* All controls start deactivated */
```

```
    b_Emergency = 0U;
}

/*
 * State Machine Procedure
 * Called by: Main.c
 * Return: void
 */
void sm_run(void) {
    if (us_State == 1U) /***** STANDBY STATE *****/ {
        if (b_On == 1U) /* If system is ON */ {
            us_State = 2U; /* Change State to IBIT */
        }
    } else /* The system check the state */ {
        if (us_State == 7U) /***** OPERATIONAL STATE *****/ {
            if (b_On == 0U) /* If system is ON */ {
                us_State = 1U; /*Change State to STANDBY*/
            }
            else {
                if (b_Operational == 1U) /* If operational ON */
{
                    operational(b_Fail); /* Call OPERATIONAL
state */
                } else {
                    shutdown(b_Fail); /* Call to Close Panel */

                    if (b_Fail == 1U) /* If fail is active */ {
                        us_State = 3U; /* Return to Fail state
*/
                    } else {
                        us_State = 4U; /* Return to READY state
*/
                    }
                }
            }

            us_State = powerMonitoring(us_State); /* Call Health
Monitoring * return: state if changed */
        }
        else if (us_State == 2U) /***** BIT STATE *****/ {
            us_State = bit(); /* Call Built-in Test */
            us_State = powerMonitoring(us_State); /* Call Health
Monitoring * return: state if changed */
        }
        else if (us_State == 3U) /***** FAIL STATE *****/ {
            b_Fail = 1U; /* Alert system to FAIL */

            if (b_On == 0U) /* If system is ON */ {
                us_State = 1U; /*Change State to STANDBY*/
            }
            if (b_Operational == 1U) {
                us_State = 7U; /* Change State to OPERATIONAL
REDUCED */
```

```
                operational_init(); /* Initialize operational
variables */
            }

            us_State = powerMonitoring(us_State); /* Call Health
Monitoring * return: state if changed */
        }
        else if (us_State == 4U) /***** READY STATE *****/ {
            if (b_On == 0U) /* If system is ON */ {
                us_State = 1U; /*Change State to STANDBY*/
            }
            if (b_Maintenance == 1U) {
                us_State = 6U; /*Change State to MAINTENANCE*/
            }
            if (b_Operational == 1U) {
                us_State = 7U; /*Change State to Operational*/
                operational_init(); /* Initialize operational
variables */
            }

            us_State = powerMonitoring(us_State); /* Call Health
Monitoring * return: state if changed */
        }
        else if (us_State == 6U) /***** MAINTENANCE STATE *****/
{
            if (b_Maintenance == 0U) /* If maintenance active */
{
                shutdown(b_Fail); /* Call to Close Panel */
                us_State = 4U; /* Change State to READY */
            }
            else {
                maintenance(); /* State MAINTENANCE */
            }

            us_State = powerMonitoring(us_State); /* Call Health
Monitoring * return: state if changed */
        }
        else {
            if (us_State == 9U) /***** EMERGENCY STATE *****/ {
                b_On = 0U; /* System Turn OFF */
                b_Fail = 0U; /* Set FAIL Alert to OFF */
                us_State = 1U; /* Change state to STANDBY */
                b_Emergency = 1U; /* Emergency Active */
                b_Operational = 0U; /* Operational OFF */
                shutdown(b_Fail); /* Shutdown System Safely */
            }
            else /***** INVALID STATE *****/ {
                b_On = 0U; /* System Turn OFF */
                b_Fail = 0U; /* Set FAIL Alert to OFF */
                us_State = 1U; /* Change state to STANDBY */
                b_Operational = 0U; /* Operational OFF */
                shutdown(b_Fail); /* Shutdown System Safely */
            }
```

```
        }
    }
}

/*****************************************************************
 * File: ibit.c
 * Author: @DaniloGraca
 *****************************************************************/

#include"monitoring.h"

/*
 * Perform the Built-in Test
 * return: Next State
 */
UINT_16 bit(void) {
    static UINT_16 us_timeCount = 0U; /* Time count */
    UINT_16 us_State; /* Storage State to return */

    if (getPower() <= 200U) /* If power less than 200w change to
FAIL state */ {
        us_State = 9U; /* Change State to EMERGENCY */
    }
    else if (getPower() <= 500U) /* If power less than 500w
change to  state */ {
        us_State = 3U; /* Change State to FAIL */
    }
    else {
        us_timeCount++; /* Iterate time counter */
        if (us_timeCount > 50U) /* Timer count to Check e change
to Operational mode*/ {
            us_State = 4U; /* Change State to READY */
            us_timeCount = 0U; /* ReStart Counter */
        }
    }
    return us_State; /* Return state updated */
}

/*****************************************************************
 * File:    operational.c
 * Author: @DaniloGraca
*****************************************************************/

#include "util.h"

static UINT_16 us_Mode; /* Store Mode */
static UINT_16 b_Panel; /* Store Panel state */
static SINT_16 ss_Position; /* Store panel position */

static void panelClose(void); /* Panel close waiting */
static void panelOpen(void); /* Panel open waiting */
static void extend(UINT_16 b_Fail); /* Change panel to open */
static void retract(UINT_16 b_Fail); /* Change panel to close */
```

```
/*
 * Initialize variables
 * Called by: stateMachine.c
 * return: void
 */
void operational_init(void) {
    b_Panel = 0U; /* Panel start Close */
    us_Mode = 1U; /* Mode start on Waiting panel */
    ss_Position = 0; /* Initial panel position ZERO / Close */
}

/*
 * Define main procedure
 * Called by: stateMachine.c
 * Param: FAIL Control
 * return: void
 */
void operational(UINT_16 FAIL) {
    if (us_Mode == 1U) /* If active mode is 1 */ {
        panelClose(); /* Call procedure panelClose */
    } else if (us_Mode == 2U) /* If active mode is 2 */ {
        extend(FAIL); /* Call procedure panelClose */
    } else if (us_Mode == 3U) /* If active mode is 3 */ {
        panelOpen(); /* Call procedure panelClose */
    }
    else /* If active mode is 4 */ {
        if (us_Mode == 4U) /* Confirm the active mode*/ {
            retract(FAIL); /* Call procedure panelClose */
        } else {
            us_Mode = 1U; /* Change to mode PANEL OPEN */
        }
    }
}

/*
 * Mode to open the panel
 * Called by: stateMachine.c
 * Param: FAIL Control
 * return: void
 */
static void extend(UINT_16 b_Fail) {
    if (b_Fail == 1U) /* If fail is active */ {
        ss_Position += 1; /* Mode EXTEND on REDUCED STATE */
    }
    else {
        ss_Position += 2; /* Mode EXTEND on OPERATIONAL STATE */
    }
    if (ss_Position > 20) /* If panel position is less than 20
*/ {
        us_Mode = 3U; /* Change to mode PANEL OPEN */
    }
}
```

```
/*
 * Mode to close the panel
 * Called by: stateMachine.c
 * Param: FAIL Control
 * return: void
 */
static void retract(UINT_16 b_Fail) {
    if (b_Fail == 1U) /* If there is a Fail */ {
        ss_Position -= 1; /* Mode RETRACT on REDUCED state */
    } else /* If this isn't a Fail */ {
        ss_Position -= 2; /*  Mode RETRACT on OPERATIONAL state
*/
    }
    if (ss_Position < 0) /* If the position of the satelite
panel is closed */ {
        us_Mode = 1U; /* Change to mode PANEL CLOSED */
    }
}

/*
 * Waiting with PANEL CLOSED
 * Called by: stateMachine.c
 * Param: FAIL Control
 * return: void
 */
static void panelClose(void) {
    if (b_Panel == 1U) /* If set panel to OPEN  */ {
        us_Mode = 2U; /* Change to mode EXTEND */
    }
}

/*
 * Mode WAITING with PANEL OPENED
 * Called by: stateMachine.c
 * Param: FAIL Control
 * return: void
 */
static void panelOpen(void) {
    if (b_Panel == 0U) /* If panel is retracted */ {
        us_Mode = 4U; /* Change to mode RETRACT */
    }
}

/*
 * Shutdown the system safely
 * This procedure is called into all cycles.
 * param: FAIL (True/False)
 * return: void
 */
void shutdown(UINT_16 FAIL) {
    while (ss_Position > 0) /* While the panel continuing closed
*/ {
```

```
        if (FAIL == 1U) /* Fail was detected */ {
            ss_Position -= 1; /* Close panel on REDUCED STATE */
        }
        else {
            ss_Position -= 2; /* Close panel on OPERATIONAL
STATE */
        }
    }
    b_Panel = 0U; /* Set panel to CLOSED */
    us_Mode = 1U; /* Set Mode to WAITING with PANEL CLOSED */
}

/*************************************************************
 * File: monitoring.c
 * Author: @DaniloGraca
*************************************************************/

#include "util.h"

static UINT_16 us_Power; /* Storage Power from panel */

/*
 * Manage power
 * return: power
 */
UINT_16 getPower(void) {
    /* us_Power (=) 100U */
    return us_Power; /* Return power */
}

/*
 * Monitoring the power from panel
 * Called by: StateMachine
 * param: State from stateMachine
 * return: State value updated
 */
UINT_16 powerMonitoring(UINT_16 state) {
    UINT_16 newState = state;

    if (getPower() < 200U) /* Testing power */ {
        newState = 9U; /* Return to EMERGENCY state */
    }
    return newState; /* Return State updated or received */
}
```

```
/*************************************************************
 * File: maintenance.c
 * Author: @DaniloGraca
 *************************************************************/

/*
 * Main procedure to maintenance
 */
void maintenance(void)
{
  /* Execute Maintenance procedures */
}

/*************************************************************
 * File:   util.h
 * Author: @DaniloGraca
 *************************************************************/

#ifndef LDRA_PORT_H
#define LDRA_PORT_H

#ifdef __cplusplus
extern "C" {
#endif

/* initialise the port */
extern void ldra_port_init(void);

/* write a zero terminated string to the port */
extern void ldra_port_write(const char* pMsg);

/* Check to see if the port has already been configured */
/* Ensures that the port is not initialised twice     */
extern int ldra_port_is_configured (void);

extern void ldra_exit (void);

#ifdef __cplusplus
}
#endif

#endif /* LDRA_PORT_H */

/*************************************************************
 * File:   util.h
 * Author: @DaniloGraca
 *************************************************************/

#ifndef UTIL_H
#define UTIL_H

#ifdef __cplusplus
extern "C" {
```

```
#endif

typedef unsigned int    UINT_16; /* Defining types to Boolean */
typedef signed int      SINT_16; /* Defining types to panel
position and time */

#ifdef __cplusplus
}
#endif

#endif /* UTIL_H */
```