Mestrado em Engenharia Informática
Estágio
Relatório Final

# ExoMars

Armando Vitor Sousa Rodrigues
avrod@student.dei.uc.pt

Orientadores:
Professor Tiago Baptista, DEI, FCTUC
Eng. Xavier Ferreira, Critical Software S.A.
Data: 1 de Julho de 2014

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# Resumo

O presente documento descreve o trabalho realizado por Armando Rodrigues no âmbito do estágio do Mestrado em Engenharia Informática, realizado na empresa Critical Software S.A.. O estágio passou pela integração na equipa de validação do *software* de bordo (abreviadamente *OBSW* em Inglês) da missão espacial *ExoMars* promovida pela Agência Espacial Europeia (ESA).

O objetivo do estágio consistiu em colaborar na validação de uma parte do *software* de bordo do *ExoMars Trace Gas Orbiter* (TGO). Nesse âmbito, os componentes validados no decorrer deste trabalho foram: *Guidance, Navigation and Control* (GNC); *Antenna Pointing Mechanism* (APM); *Solar Array Deployment Mechanism* (SADM); *Thermal Regulation* (TR); *Entry Descendent Module* (EDM).

O *software* dos componentes anteriormente referidos foi integralmente implementado e validado pela Critical Software. Outros componentes fazem também parte do OBSW do TGO, tal como o *System Management Software* (SMS), no entanto esses foram desenvolvidos e validados por outra empresa que não a Critical Software.

O desenvolvimento de *software* de um satélite é considerado "*mission critical*", já que uma pequena falha eventualmente fácil de corrigir em terra, após o lançamento pode levar à perda permanente do satélite. A validação garante que o *software* desenvolvido cumpre todos os requisitos e o seu comportamento corresponde exatamente ao que é especificado de forma a minimizar a probabilidade do *software* ser lançado com defeitos.

De forma a atingir os objetivos, foi necessário elaborar um estudo sobre validação de *software*, conceitos de engenharia aeroespacial e o funcionamento do *ExoMars TGO*. De seguida foram especificados, implementados e executados com sucesso vários testes, que ajudaram a descobrir alguns defeitos no software.

O objetivo proposto inicialmente foi cumprido, até ultrapassado. Para além, da validação prevista inicialmente, foi ainda, desenvolvido uma biblioteca em Java para auxiliar a validação sobre o protocolo MIL-STD-1553b.

# Palavras-Chave

Software, Validação, Testes, *ExoMars*, Critical Software, Agência Espacial Europeia, ESA, Waterfall, black-box.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Tabela de Acrónimos

| Acrónimo | Descrição |
|---|---|
| APM | Antenna Pointing Mechanism |
| EDM | Entry Descendent Module |
| GNC | Guidance, Navigation and Control |
| PUS | Packet Utilization Standard |
| SADM | Solar Array Deployment Mechanism |
| SMUIO | Spacecraft Management Unit Input/Output |
| SRS | Software Requirements Specification |
| SVF | Software Validation Facility |
| SVTExec | Software Validation Test Execution |
| SVTProc | Software Validation Test Procedures |
| SVTR | Software Validation Test Report |
| SVTS | Software Validation Testing Specifications |
| TGO | Trace Gas Orbiter |
| TR | Thermal Regulation |

**Tabela 1 - Tabela de Acrónimos [1]**

# Capítulo 1
# Introdução

O presente documento constitui o relatório desenvolvido no âmbito do estágio do Mestrado de Engenharia Informática do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra, no ano letivo 2013/2014. O estágio consistiu na validação do *software* do satélite *ExoMars* com uma duração de dois semestres. Este documento retrata todo o trabalho e aprendizagem adquirida ao longo dos dois semestres. O estágio teve inicio a 10 de Setembro de 2013 e terminou a 27 Junho de 2014. Todo o trabalho foi desenvolvido nas instalações da empresa Critical Software S.A., em Coimbra.

O estágio foi orientado pelo Prof. Tiago Baptista, por parte da Universidade de Coimbra e pelo Eng. Xavier Ferreira, da parte da Critical Software S.A..

## 1.1. *ExoMars*

As missões *ExoMars* fazem parte do programa Aurora da Agencia Espacial Europeia (ESA)[2]. O objetivo do programa Aurora é formular e implementar um plano europeu de longo prazo para a exploração robótica e humana dos corpos do sistema solar esperando conseguir descobrir vestígios de vida. As missões *ExoMars* são as primeiras do programa Aurora, e têm como objetivo investigar o ambiente marciano, abrindo o caminho para uma futura missão de retorno de amostras de Marte em 2020. Duas missões estão previstas dentro do programa *ExoMars*: a primeira é a *ExoMars Trace Gas Orbiter* (TGO)[3], e a segunda é a *ExoMars Rover*.

Na Figura 1 está uma representação do *ExoMars TGO* (à esquerda) e do *ExoMars Rover* (à direita).

**Figura 1 - Imagem representativa do *ExoMars TGO* e *ExoMars Rover* [2]**

Os objetivos tecnológicos das missões *ExoMars* são desenvolver e qualificar [2]:

- Estudo dos gases da atmosfera de Marte;
- Entrada, descida e aterragem de uma carga na superfície de Marte;
- Mover-se na superfície com um *rover*;
- O acesso ao subsolo para a aquisição de amostras;
- Aquisição da amostra, preparação, distribuição e análise;

A primeira missão, cujo lançamento está previsto ser em 2016, pretende chegar à orbita de Marte e estudar a atmosfera durante um ano marciano. Esta sonda também contém uma carga para testar a entrada, descida e aterragem em Marte. Os principais objetivos desta missão consistem em procurar evidências de metano e outros gases atmosféricos que poderão ser assinaturas de processos biológicos ou geológicos ativos, assim como testar tecnologias-chave na preparação de posteriores missões a Marte.

O TGO vai transportar uma carga útil científica capaz de abordar a deteção e caracterização dos gases na atmosfera marciana. Desde a sua chegada a ambiente marciano, os instrumentos a bordo do TGO terão a capacidade de detetar uma ampla gama de gases atmosféricos (como o metano, vapor de água, dióxido de nitrogênio, acetileno), com uma maior precisão em comparação com as medições anteriores (de três ordens de grandeza). As medições realizadas pelo TGO irão fornecer evidências sobre a localização e as fontes desses gases, que permitirá definir locais de aterragem para futuras missões.

Para além disso, o TGO irá transportar a *Entry Descendent Module* (EDM) na viagem Terra-Marte.

Na primeira missão do *ExoMars* o EDM não irá transportar o *Rover*, em vez disso irá transportar um objeto com o mesmo peso e com as mesmas dimensões que o *Rover*. Isto deve-se à necessidade da Agência Espacial Europeia (ESA) testar a entrada, descida e aterragem na superfície do planeta.

Após o lançamento do *ExoMars TGO*, o satélite executará uma rota pré-programada até Marte. Ao chegar, o satélite ficará na sua orbita. Quando este estiver a orbitar em Marte, irá lançar o EDM para a superfície. Após o lançamento do EDM o TGO vai receber e retransmitir para terra os dados recebidos do EDM em ondas de alta frequência (UHF).

## 1.2. *ExoMars* na Critical Software

Na Figura 2 é possível observar uma representação de alto nível da arquitetura do *software* de bordo do *ExoMars* TGO. Nesta Figura os módulos a amarelo, *Hardware Dependent Software* (HDSW) e *Board Support Package* (BSP) incluem todo o *software* dependente das plataformas de hardware, como os drivers e o *boot software*. Estes equipamentos são disponibilizados pelos fornecedores do *hardware*. Os módulos a azul, *Input Output Services* (IOS) e o *Real Time Kernel* (RTK), são os responsáveis pelo serviço de comunicação e pelo serviço de agendamento de todo o *software*.

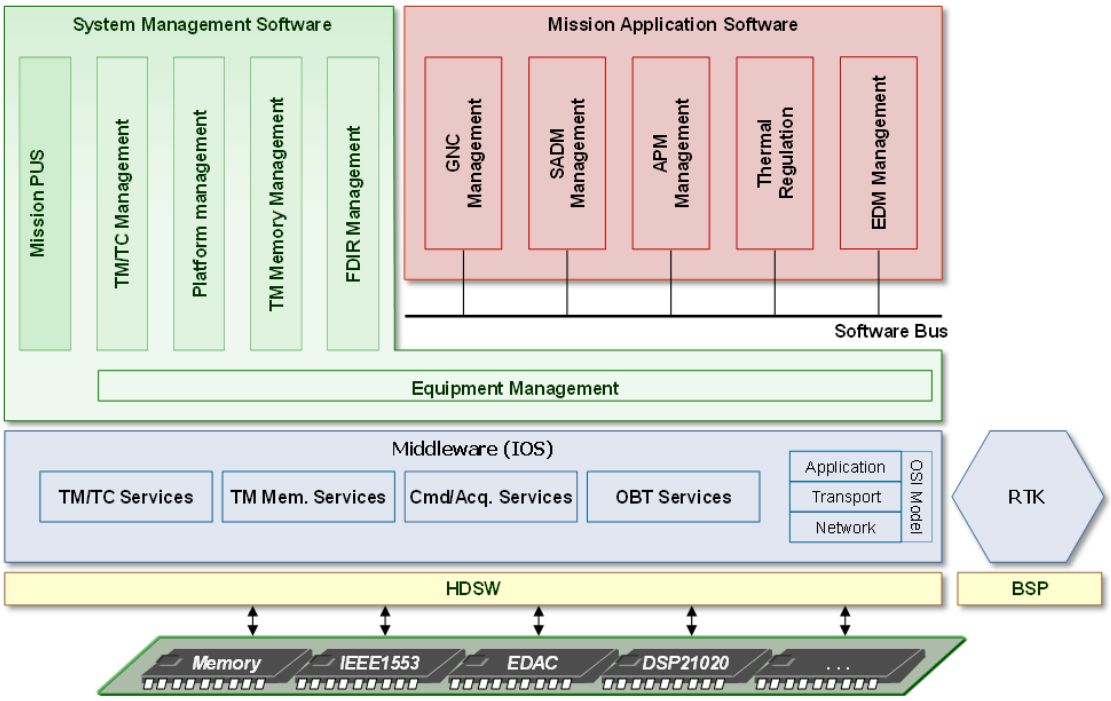



**Figura 2 - Arquitetura do *ExoMars* TGO [2]**

Ilustrado a vermelho é possível observar o *Mission Application Software* (MAS). Este módulo é dividido em cinco componentes [2]:

- ***Guidance, Navigation and Control* (GNC) *Management*** - Fornece os serviços necessários para gerir a orientação, a navegação e os modos de controlo [4].
- ***Solar Array Drive Motor* (SADM) *Management*** - Fornece os serviços necessários para gerir o Solar Array Drive Motor [5].
- ***Antenna Pointing Mechanism* (APM) *Management*** - Fornece os serviços necessários para gerir o Antenna Pointing Mechanism [6].
- ***Thermal Regulation* (TR) *Management*** - Fornece os serviços necessários à regulação térmica (processamento de valores térmicos e controlo os aquecedores.
- ***Entry Descendent Module* (EDM) *Management*** - Fornece os serviços necessários para gerir o equipamento EDM, que é considerado o *payload* do satélite.

A Critical Software é a responsável pelo desenvolvimento e validação do *software* destes cinco componentes.

Por fim a verde é possível observar o *System Management Software* (SMS). Este módulo é implementado e validado pela Thales Alenia Space. O *System Management Software* (SMS), é dividido em seis componentes principais, a saber:

- ***Packet Utilization Standard* (PUS)** - Fornece o serviço PUS conforme especificado pela norma ECSS-70-42B[7][8].
- ***TCTM Management* (TCTM Mgmt)** - Fornece os serviços necessários à gestão de telemetria e telecomandos.
- ***TM Memory Management* (TMMem Mgmt)** - Fornece as funções necessários para a gestão eficiente da memória.
- ***Platform Management* (Platform Mgmt)** - Fornece a maioria dos serviços de tratamento dos dados para todos os outros componentes.

- ***Failure Detection, Isolation and Recovery*** **(FDIR)** - Fornece as funcionalidades necessárias à deteção e a recuperação de falhas.
- ***Equipment Management*** **(Equipment Mgmt)** - Implementa as funções necessárias à gestão das plataformas, incluindo as unidades complexas.

## 1.3. Objetivos do Estágio

O objetivo principal deste estágio foi participar na validação do *software* do *ExoMars Trace Gas Orbiter* (TGO). Para tal foi necessário:

- Adquirir conhecimento sobre sistemas Aeroespaciais;
- Adquirir conhecimento sobre validação de *software*;
- Estudar os requisitos de cada componente;
- Especificar diversos testes, com objetivo de cobrirem todos os requisitos dos componentes;
- Implementar os testes especificados;
- Executar os testes no simulador de tempo real presente no ambiente de validação do *software*;
- Identificar e reportar eventuais falhas às equipas de desenvolvimento;
- Voltar a executar os testes com falhas detetadas a fim de validar as correções.
- Refletir nas evoluções dos documentos de requisitos (requisitos novos, modificados, apagados) nas especificações de testes existentes e na sua implementação e voltar a executar os novos testes e os testes modificados.
- Manter o estado de cada teste atualizado de forma a ser possível aferir o estado da campanha de testes para cada componente.

## 1.4. Estrutura do documento

Esta secção tem como objetivo possibilitar ao leitor uma breve descrição dos capítulos existentes e a estrutura do documento apresentado.

No **capítulo 2**, Estado da Arte, estuda-se as principais técnicas de validação de *software*

No **capítulo 3**, Planeamento e Metodologia, dá-se a conhecer a equipa e todo o planeamento do projeto. Também é referido o ambiente de validação, equipamentos e ferramentas utilizadas.

No **capítulo 4**, *ExoMars Trace Gas Orbiter* (TGO), é descrita uma introdução aos sistemas aeroespaciais e ao funcionamento do *ExoMars TGO*.

No **capítulo 5**, Processo de Validação, é descrito o processo de validação a utilizar para a validação do *software* do *ExoMars TGO*.

No **capítulo 6**, MIL-STD-1553b, é descrito o funcionamento do Mil-STD 1553b e todo o trabalho realizado.

No **capítulo 7**, Resultados, é feito um resumo do trabalho realizado e os resultados obtidos.

No **capítulo 8**, Conclusão, destaca-se os objetivos atingidos neste projeto de estágio e é feito um balanço dos conhecimentos e competências adquiridas pelo estagiário.

# Capítulo 2
# Estado da Arte

O presente estágio diz respeito à validação do *software* do *ExoMars Trace Gas Orbiter* (TGO). A validação do *software* consiste em garantir que este cumpre todos os requisitos especificados. Para conseguirmos garantir o cumprimento dos requisitos, é necessário testar o mesmo. Este capítulo começa com uma introdução ao processo de desenvolvimento de *software*, descrevendo de seguida algumas das técnicas de teste.

## 2.1 Processo de desenvolvimento de *software*

O processo de desenvolvimento do *software* pode ter diferentes abordagens, consoante as suas características e necessidades. Dentro das mais utilizadas, encontra-se o *Waterfall* e os modelos ágeis. Inserido no âmbito dos modelos ágeis, apenas irá ser abordado o *Scrum* e o *Test-driven*.

### 2.1.1 *Waterfall*

O modelo *Waterfall* é um modelo de desenvolvimento sequencial, onde é necessário seguir as ordens das tarefas, pois muitas vezes estas são dependentes das suas antecessoras.

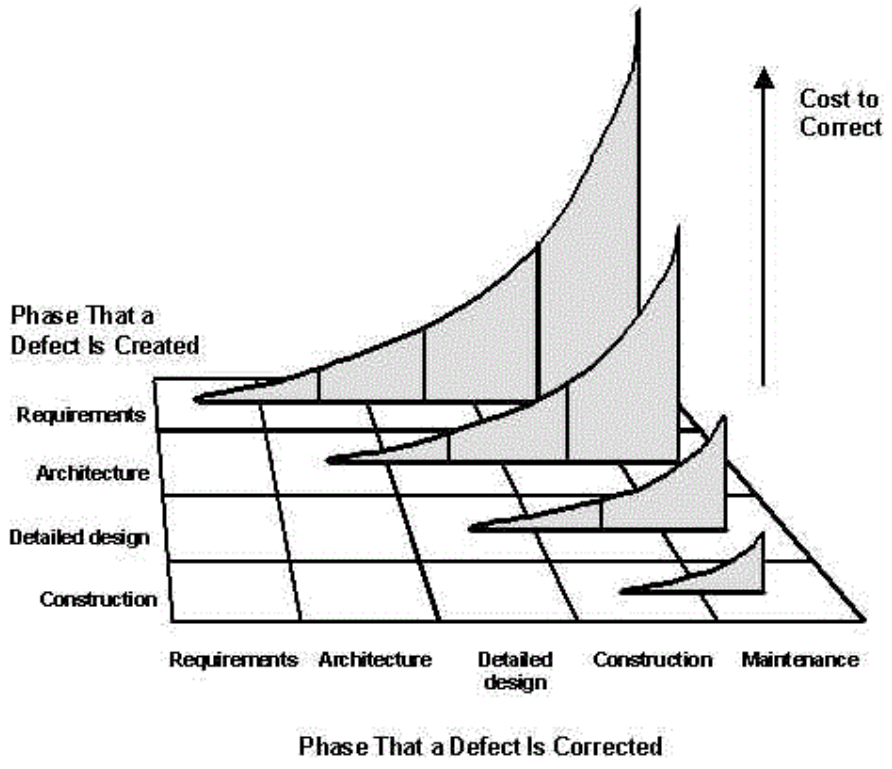



**Figura 3 - Custo de correção de um defeito[9]**

O principal objetivo deste modelo, é corrigir os erros o mais cedo possível, pois quanto mais tarde um erro for detetado mais tempo e dinheiro será necessário para o corrigir. Como McConnell demonstra e está representado na Figura 3, um erro encontrado nas fases iniciais (como a análise e especificação de requisitos ou a especificação do design) é menos dispendioso em dinheiro, em esforço e em tempo, comparado com um erro encontrando mais tarde no processo [9]. Esta metodologia de desenvolvimento é mais adequada para

projetos onde é possível se realizar uma especificação detalhada dos requisitos no início do projeto.

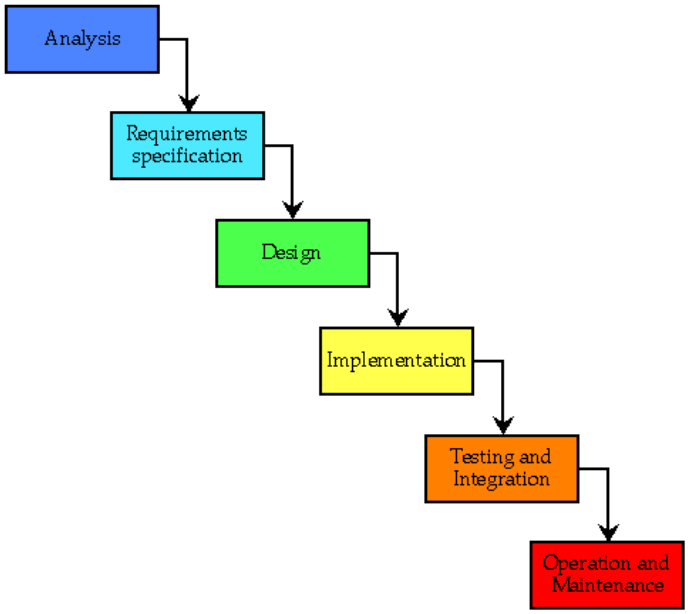



**Figura 4 - Modelo *Waterfall* tradicional[10]**

Na Figura 4 estão representadas as várias fases do processo de desenvolvimento do *software* pelo modelo *Waterfall*. Como se pode observar na figura existem seis fases:

- Análise: Investigação dos requisitos do cliente;
- Especificação: Especificação dos requisitos;
- Design: Desenho e especificação de uma solução que cumpra os requisitos;
- Implementação: Fase onde é escrito o código, documentação do código e é realizado alguns testes (normalmente testes unitários);
- Teste e integração: Verificação através de testes do cumprimento dos requisitos especificados;
- Manutenção: Correção de pequenos bugs não descobertos anteriormente;

No caso do *Waterfall* a validação concentra-se na fase "teste e integração", mas também pode estar presente nas restantes fases do processo de desenvolvimento do *software*.

### 2.1.2 *Scrum*

O *Scrum* é metodologia de gestão de processos iterativa e incremental com base num modelo ágil. Um dos princípios chaves das metodologias ágeis é a faciliade de lidar com mudanças constantes de opinião por parte dos clientes. Este processo funciona por pequenos sprints e reuniões diárias com toda a equipa de trabalho, para avaliar o progresso do projeto e debater em equipa pequenos problemas encontrados.

Como se pode observar na Figura 5, toda a lógica de *Scrum* tem início com uma visão, uma ideia de produto, que ganha forma através de um *Product Backlog*. Este está em constante evolução sendo revisto, reavaliado e atualizado a cada sprint pelo *Product Owner*.

De acordo com Mitch Lacey [11], após a sua definição inicial, um *Product Backlog* é concretizado em ciclos de desenvolvimento (*sprints*) que normalmente duram entre duas a

quatro semanas, onde todo o trabalho (*Sprint Backlog*) é planeado e executado com o pressuposto de no final de cada ciclo entregar uma ou mais componentes funcionais do produto.

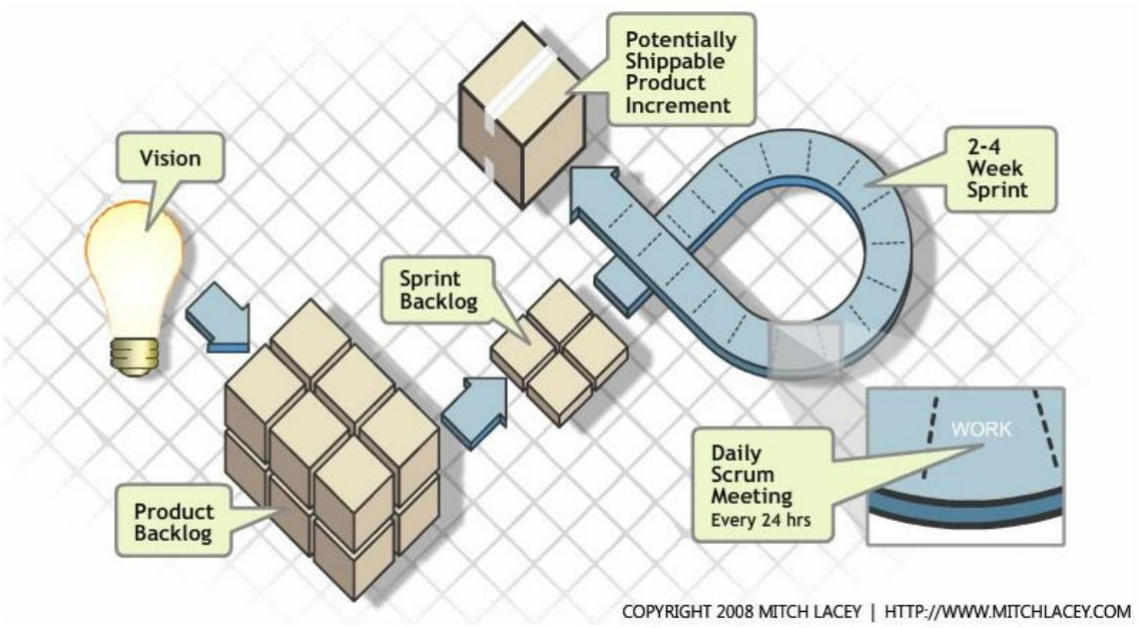



**Figura 5 - Metodologia de desenvolvimento ágil *Scrum* [11]**

Ao longo de todo o ciclo de desenvolvimento existe diariamente uma sessão, a *Daily Scrum Meeting*, onde a equipa se reúne para avaliar o ponto da situação. No final de cada ciclo, o *Product Owner* avalia então todo o trabalho realizado pela equipa na sessão de *Review*.

### 2.1.3 *Test Driven*

O *Test Driven Development* (TDD) é uma técnica de desenvolvimento de *software* orientado a testes, caracterizada por serem escritos os testes antes do código. Como representado na Figura 6, o *Test Driven Development* é um processo cíclico que funciona da seguinte forma: primeiro é escrito um teste automatizado para uma funcionalidade ou componente a ser implementado. Após a conclusão do teste, é escrito o código necessário para passar no teste. Por fim, caso seja necessário é efectuada uma limpeza e/ou optimização do código.

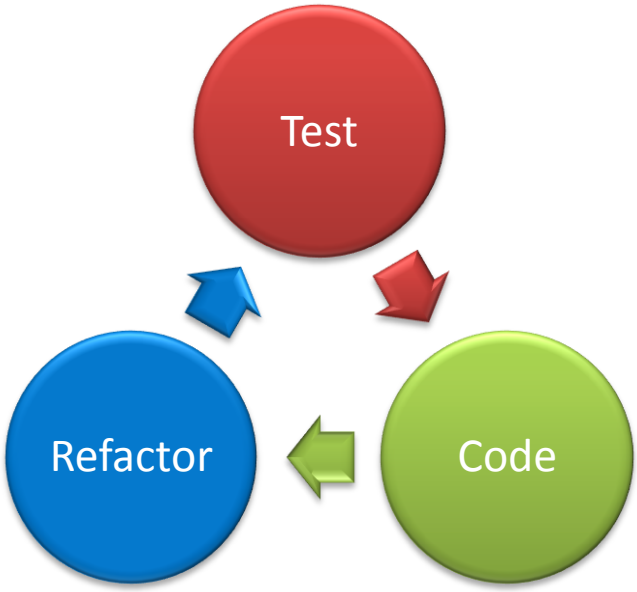



**Figura 6 - Ciclo de Desenvolvimento em *Test Driven Development***

Uma vez que os testes são automatizados e são escritos antes da implementação, no decorrer da mesma, pode-se executar várias vezes os testes até a implementação passar nos mesmos. Caso os testes sejam bem escritos, conseguimos garantir que se a implementação passar ela desempenha o comportamento desejado.

Esta técnica faz com que o programador deixe de ser influenciado pelo código já feito, e programa a pensar no comportamento em vez da implementação.

## 2.2 Categorias de teste de *software*

O *software* pode ser testado por verificação ou validação. A verificação consiste em garantir que o *software* cumpre os requisitos definidos pelo cliente. Por sua vez a validação consiste em garantir que o comportamento do *software* corresponde ao esperado pelo cliente. Tendo como exemplo uma fábrica de carros, verificar seria garantir que as várias peças que constituem o veículo são as corretas, não contêm defeitos, e são montadas corretamente. Já validar seria por exemplo, garantir que ao carregar nos travões o carro trava como é esperado.

Existem várias técnicas para teste de *software*, as principais técnicas de teste são *Black Box* e *White Box*. Qualquer uma destas abordagens pode ser aplicada, em princípio, durante qualquer estágio do processo de testes, contudo cada uma delas é preferencialmente aplicável a determinados tipos de componentes e realizáveis por equipas distintas.

### 2.2.1 Testes estáticos e dinâmicos

Nos testes estáticos não é necessário executar o software, pode-se por exemplo fazer uma análise ao código. Contudo, nos testes dinâmicos é necessário executar e no final é gerado um relatório com os resultados. Os resultados devem ~~de~~ ser examinados para verificar se o programa operou de acordo com o esperado.

Os testes estáticos são os mais usados na verificação de *software*, enquanto os dinâmicos são mais usados na validação. No entanto, ambos podem ser usados nas diferentes etapas do processo de desenvolvimento de *software*.

### 2.2.2 *Black-Box e White-Box*

Na abordagem *Black-Box* o *software* é testado sem a visualiação do código. Os casos de testes são derivados a partir da especificação do sistema ou componente a ser testado. O sistema é visto como uma caixa fechada e o seu comportamento apenas pode ser derivado através do estudo dos possíveis valores de entrada e dos valores de saída relacionados. Esta abordagem utiliza testes dinâmicos, sendo melhor aplicada sob componentes de sistema e, realizada por uma equipa de testes.

Na abordagem White-Box a pessoa que compete testar, pode analisar o código e usar o conhecimento da estrutura do componente para derivar os casos de testes. Esta abordagem utiliza normalmente testes estáticos, é sendo melhor aplicada a componentes individuais ou a conjuntos de componentes dependentes e realizada pela equipa de desenvolvimento.

## 2.3 Fases de teste de *software*

Devido à complexidade dos sistemas a atividade de testes deve ser feita ao longo de diferentes estágios. Deste modo, processo de testes mais utilizado é composto por quatro

estágios, como o ilustrado na Figura 7. O processo é iterativo, sendo que a informação produzida em cada estágio, poderá fluir entre estágios adjacentes.

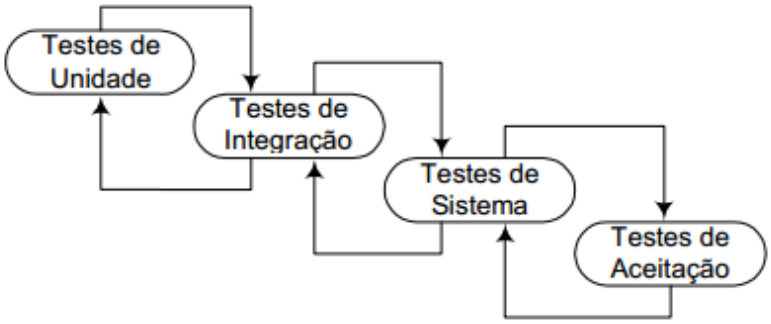



**Figura 7 - Visão geral do processo de testes [12]**

Os principais testes utilizados [12] são: Testes de Unidade, Testes Integração, Testes de Sistema e Testes de Aceitação.

### 2.3.1 Testes de Unidade

O objetivo é encontrar falhas de funcionamento numa pequena parte do *software*. Os testes de unidade fazem parte do processo de desenvolvimento do *software* e são da responsabilidade dos programadores que estiveram a desenvolver o componente. Normalmente testam individualmente cada função, mas podem variar dependendo do paradigma de programação utilizado.

### 2.3.2 Testes Integração

Este tipo têm como objetivo de testar a integração dos vários componentes de um sistema. Acontece que, por norma em grandes sistemas, existe um problema na integração das interfaces entre os módulos. Assim, estes testes devem concentrar-se no exercício rigoroso de tais interfaces para deteção de possíveis erros.

### 2.3.3 Testes de Sistema

Os testes de sistema realizam-se ao sistema completo, para verificar que este cumpre todos os requisitos e comporta-se como esperado. Tendo como objetivo, detectar falhas resultantes das interacções entre componentes de sistema.

### 2.3.4 Testes de Aceitação

Este é o estágio final do processo de testes antes do sistema ser aceite para uso operacional. O sistema é testado com dados fornecidos pelo utilizador final, ao contrário de dados fictícios ou simulados. Os testes de aceitação revelam principalmente erros e omissões na definição dos requisitos, pois através do uso de dados reais o sistema é exercitado de formas variadas.

# Capítulo 3
# Planeamento e Metodologia

Neste capítulo é descrito o planeamento durante o período de estágio e especificada a metodologia de trabalho. Também é especificada as equipas de trabalho do projecto e sua constituição, assim como o ambiente de trabalho e respetivas ferramentas e equipamentos.

## 3.1. Equipa

O projeto *ExoMars* é constituído por duas equipas: equipa de desenvolvimento liderada por João Brito, e a equipa de validação liderada por Xavier Ferreira (Orientador do estágio).

Este estágio insere-se na equipa de validação, equipa esta, constituída por Xavier Ferreira, Armando Rodrigues, Hugo Almeida, João Teixeira e Gualter Fontes em Coimbra e Tiago Saraiva e Filipe Pedrosa em Cannes (França). Para além da equipa de trabalho em Coimbra, existe um contacto regular com a equipa da Thales Alenia Space para dúvidas e escarecimentos da documentação ou do funcionamento do Hardware.

A distribuição do trabalho é feita através de uma folha de controlo. Na Figura 8 podemos ver uma perspetiva dessa mesma folha. Cada linha da tabela diz respeito a um teste e é atribuído a uma pessoa. É também registado o estado do teste e dos comentários quando necessário (Lista dos pontos bloqueantes, adicionados ligações para eventuais problemas encontrados, etc).

| | A | B | C | D | E | F | I | J | K | L | M | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Component | SW Mode | Tsk | Tas_Sts | ML | RUN | Spec ID | Spec Title | Tester | Execution Status | Execution Result | Comments |
| 30 | GNC | SBM | ANA | ACTIVE | ML-TRB | 1 | AN-GNC-SBM-INIT | GNC SBM Initialisations | jmteixeira | EXECUTED | OK | |
| 31 | GNC | Generic | ANA | ACTIVE | ML-TRB | 1 | AN-GNC-TASK | GNC Task | hjalmeida | PLANNED | PENDING | |
| 42 | APM | | TST | ACTIVE | ML-TRB | 4 | VT-APM-AJ-PARAMS | APM AJ Parameters | cxferreira | EXECUTED | OK | |
| 48 | APM | | TST | ACTIVE | ML-TRB | 4 | VT-APM-DAT-PARAMS | APM DAT Parameters | hjalmeida | EXECUTED | PENDING | |
| 50 | APM | | TST | ACTIVE | ML-TRB | 2 | VT-APM-MANUAL-LONG-RUN | APM Manual Mode Long Run | fjdpedrosa | EXECUTED | PENDING | |
| 55 | APM | | TST | ACTIVE | ML-TRB | 4 | VT-APM-MODEFUNC-INIT | APM Mode and Function Initialisation | hjalmeida | EXECUTED | PENDING | |
| 57 | APM | Generic | TST | ACTIVE | ML-TRB | 2 | VT-APM-MODE-TRANSITIONS | APM Mode Transitions | arodrigues | EXECUTED | OK | |
| 63 | APM | | TST | ACTIVE | ML-TRB | 4 | VT-APM-TC-ERR | APM TeleCommand Checks | arodrigues | EXECUTED | PENDING | |
| 65 | GNC | AEBM | TST | ACTIVE | ML-TRB | 2 | VT-GNC-AEBM-INIT | Aero Braking Mode Initialisations | hjalmeida | EXECUTED | PENDING | |
| 66 | GNC | AEBM | TST | ACTIVE | ML-TRB | 1 | VT-GNC-AEBM-LONG-RUN | GNC AEBM Long Run | gfsilva | PLANNED | PENDING | |
| 69 | GNC | | TST | ACTIVE | ML-TRB | 3 | VT-GNC-AEBM-PASS-INIT | AEBM Pass initialisation | hjalmeida | EXECUTED | PENDING | |
| 71 | GNC | AEBM | TST | ACTIVE | ML-TRB | 2 | VT-GNC-AEBM-PASS-SUBMODE | AEBM Pass Submode | hjalmeida | EXECUTED | PENDING | |
| 74 | GNC | AEBM | TST | ACTIVE | ML-TRB | 3 | VT-GNC-AEBM-PTE-PROCESSING | Aero Braking Mode PTE Processing | hjalmeida | EXECUTED | PENDING | |
| 77 | GNC | | TST | ACTIVE | ML-TRB | 3 | VT-GNC-AEBM-RD-INIT | AEBM Rate Damping Initialisation | hjalmeida | EXECUTED | PENDING | |
| 79 | GNC | AEBM | TST | ACTIVE | ML-TRB | 2 | VT-GNC-AEBM-RD-SUBMODE | AEBM Rate Damping Submode | jmteixeira | EXECUTED | PENDING | |
| 83 | GNC | | TST | ACTIVE | ML-TRB | 3 | VT-GNC-AEBM-SUBMODE | AEBM Submodes | jmteixeira | EXECUTED | PENDING | |

**Figura 8 - Folha de controlo para as tarefas de validação.**

Esta folha de controlo é na realidade um ficheiro Excel. Uma possível melhoria seria utilizar uma ferramenta que permitisse fazer a mesma gestão dos testes, mas com a capacidade de fornecer estatísticas sobre todo o progresso dos testes. Estes dados poderiam vir a ser uma mais-valia para futuros projetos similares.

## 3.2. Metodologia de Desenvolvimento

No que diz respeito à metodologia de desenvolvimento, o projeto *ExoMars* utiliza uma metodologia *Waterfall*. Neste projeto é feita uma junção entre o *Waterfall* e o *Test Driven*. Como se pode verificar na Figura 9, a equipa de validação começa a trabalhar ao mesmo tempo que a equipa de desenvolvimento.

Tendo em conta as características deste projeto, não se justifica utilizar um modelo ágil. Uma das razões é a estabilidade dos requisitos, estes foram definidos no início do projeto e não era esperado alterações significativas. O mesmo veio-se a comprovar mais tarde.

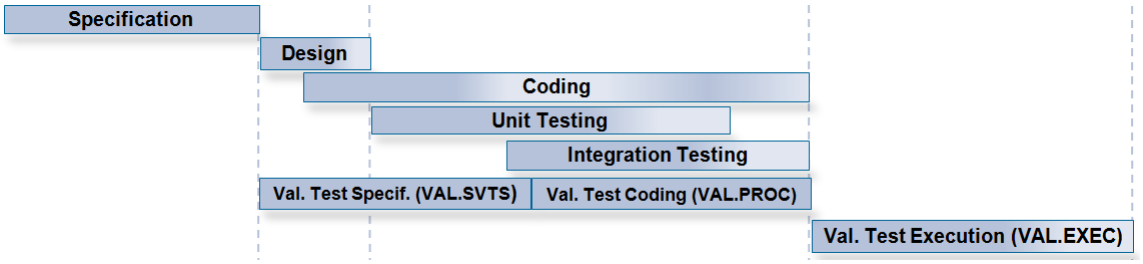



**Figura 9 - Diagrama do projeto[13]**

Durante a codificação, a equipa de desenvolvimento faz testes de unidade, testes de integração e de revisões de código. Na atividade de *coding*, também é feita revisão de código. Todo este processo é realizado segundo a técnica *White-Box* e os testes são na maioria estáticos.

Todo o processo de validação é realizado utilizando *Black-Box* e testes dinâmicos.

## 3.3. Ambiente de validação

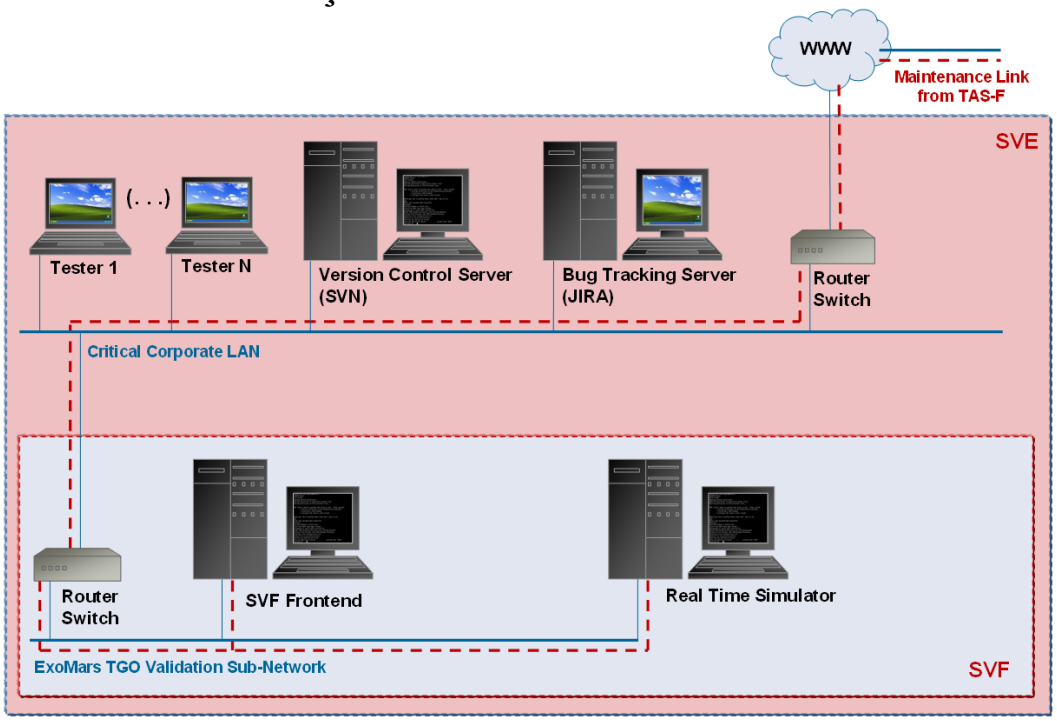



**Figura 10 - Software Validation Environment [2]**

Este subcapítulo identifica os equipamentos e as ferramentas que serão utilizadas nas diferentes atividades da validação do *ExoMars TGO* durante este estágio. Na Figura 10 está representado o *Software Validation Environment* (SVE) [6].

A *Software Validation Facility* (SVF) é constituída pela *SVF Front End* e pelo *Real Time Simulator*, é apenas utilizado para correr os testes. O restante é utilizado em todas as atividades da validação.

### 3.3.1. Equipamentos

Como podemos observar na Figura 10, os equipamentos utilizados na validação são:

| Nome | Descrição |
|---|---|
| Servidor de controlo de versões (SVN) | Servidor com controlo de acesso por autenticação, onde os membros da equipa de validação submetem o seu trabalho. Este servidor evita a perda de informações, como também a possível consulta de versões anteriores do trabalho. |
| Servidor de registo e acompanhamento de erros/problemas | Servidor com controlo de acesso por autenticação, onde é feito o registo dos diversos erros encontrados durante o processo de validação. Estes erros vão ser resolvidos mais tarde pela pessoa responsável. |
| Portáteis | Todo o trabalho dos membros da equipa de validação será executado a partir do seu portátil. |
| *Real Time Simulator* (RTS) | Esta máquina contém o simulador, como tal, é a máquina mais potente. Permite a simulação dos equipamentos do satélite, e as comunicações entre os equipamentos. |
| *SVF Front End* | É neste servidor que são compilados os testes, os quais ficam a aguardar sequencialmente para serem executados no *Real Time Simulator* (RTS) e são analisados após serem executados. |

**Tabela 2 - Lista de Equipamentos**

### 3.3.2. Ferramentas

| Ferramenta | Utilização | Atividade |
|---|---|---|
| Intradoc | Documentação do projeto | Especificação |
| TortoiseSVN | Controlo de versões dos diversos ficheiros e documentos | Especificação, Codificação, Execução |
| Notepad++ | Escrita de código e visualização dos resultados da simulação | Codificação, Execução |
| Enterprise Architect | Escrita das especificações dos testes | Especificação |
| Putty + Xming | Acesso remoto por ssh a SVF | Codificação, Execução |
| JIRA | Ferramenta de registo de erros usada pela Critical Software e pela Thales Alenia Space | Especificação, Codificação, Execução |
| MS Word | Escrita das SVTS | Especificação |
| MS Excel | Visualizar resultados da simulação | Execução |
| Eclipse | Editor e compilador | Mil-STD 1553b |

**Tabela 3 - Lista de Ferramentas**

## 3.4. Planeamento

Apesar de se tratar de um estágio de mestrado, o *ExoMars* tinha um planeamento definido entre a Critical Software e a Thales Alenia Space. Desta forma, o presente estágio ficou condicionado por essa planificação.

O estágio foi dividido em dois semestres, o primeiro decorreu entre Setembro de 2013 e Janeiro de 2014, o segundo decorreu de Fevereiro a Junho de 2014. No primeiro semestre a alocação minima obrigatória era de 40%, mas devido à complexidade e diferenciação deste projecto em relação à maioria dos projctos de estágio, decidiu-se por uma alocação na ordem dos 80% no primeiro semestre e 100% no segundo. No Apêndice A está repressentado o Gantt do estágio. Uma vez que este projecto não dependia apenas da Critical Software, este gantt foi elaborado numa estimativa de três pontos, sendo as datas representadas optimistas. Ora, de acordo com a perspetiva otimista o projecto terminaria na segunda semana de Maio, na pessimista na segunda semana de Junho. Apesar de vários atrasos por parte da Thales (responsável pela especificação dos requisitos de todos os componentes, pela integração e geração de cada binário do *software* de bordo e pela manutenção do ambiente de validação), o planeamento foi cumprido, visto que terminou no início do mês de Junho. O restante tempo ficou reservado à elaboração escrita da dissertação final.

### 3.4.1. Primeiro Semestre

O primeiro semestre teve início com o estudo de três dos módulos do satélite (SADM [10], APM [11] e GNC [9]), e análise dos requisitos definidos pela Thales Alenia Space para cada um desses módulos. Estes três módulos (e outros da responsabilidade da Thales e não referidos aqui) fazem parte da fase V1 do desenvolvimento e validação (o desenvolvimento do software de bordo é dividido em 3 fases de acordo com as dependências com o hardware e das dependências do vários componentes entre si) e por isso foram os primeiros a ser trabalhados.

Após concluir essa análise, foi necessário a criação dos casos de teste, de forma a cobrir integralmente os requisitos. Posteriormente passou-se a implementação dos mesmos. Nesta duas fases foi necessário um grande acompanhamento pois era necessário conhecer bem as características e as funcionalidades inerentes ao ambiente de validação, cuja implementação e o uso é específico à validação de satélites no âmbito da Thales Alenia Space.

Depois destes três módulos serem devidamente codificados, e porque a Thales Alenia Space estava atrasada a acabar a sua parte da fase V1 e nas entregas para a V2 (especificação de requisitos e base de dados do satélite para os componentes TR e EDM), esteve-se algum tempo a ajudar a Thales Alenia Space a especificar, codificar e a executar os testes do componente *Spacecraft Management Unit Input/Output* (SMUIO), um subcomponente da camada *System Management Software* (SMS). Este componente é da responsabilidade Thales, mas devido ao atraso por parte da Thales nos seus projetos, foi acordado entre as duas entidades de entregar a validação deste componente à Critical Software, evitando a dispersão de recursos para outros projetos, ajudando a Thales a recuperar o atraso na V1 e fazendo assim um compasso de espera ativa para superar o atraso nas dependências da V2.

Pelo exposto, o referido acordo desencadeou algumas alterações no planeamento inicial deste estágio. Além disso, foi necessário incorporar uma nova metodologia de codificação de testes, que já tinha sido introduzida nos componentes da Thales, mas ainda não tinha sido implementada nos componentes da responsabilidade da Critical Software (a introdução de uma camada adicional para a codificação e execução de testes no ambiente de validação. Essa nova camada, o VERTIGO permite a codificação de testes em java, em detrimento da

linguagem especifica baseada em C, e fornece um ambiente integrado de codificação e execução de testes baseado plataforma do eclipse).

O componente SMUIO é um componente responsável por gerir as aquisições e comandos (IO do termo SMUIO) da placa mãe (nela está o processador onde correr o software de bordo) do satélite (*Spacecraft Management Unit/SMU* do termo SMUIO). A placa mãe do satélite comunica com os restantes equipamentos do satélite na sua maioria através de um barramento de especificação militar, o Mil-STD 1553b (Capítulo 4.1.5). Uma das funções do presente estágio, foi também, a de realização de uma libraria para tornar mais simples e disponibilizar mais meios de verificar a passagem das mensagens Mil-STD 1553b referidas nos requisitos (e portanto presentes nos também nos testes como verificações a fazer) neste e outros componentes validados usando o VERTIGO.

Na tabela seguinte estão representadas as tarefas realizadas no primeiro semestre, assim como as respetivas datas de inicío e fim, bem como a duração em dias úteis.

| | Início | Fim | Duração(dias úteis) |
|---|---|---|---|
| Estudo dos módulos SADM, APM e GNC | 10/09/2013 | 21/10/2013 | 30 |
| Especificação dos testes do SADM | 16/09/2013 | 17/09/2013 | 2 |
| Especificação dos testes do APM | 18/09/2013 | 19/09/2013 | 2 |
| Especificação dos testes do GNC | 20/09/2013 | 02/10/2013 | 9 |
| Codificação dos testes do SADM | 03/10/2013 | 08/10/2013 | 4 |
| Codificação dos testes do APM | 09/10/2013 | 14/10/2013 | 4 |
| Codificação dos testes do GNC | 15/10/2013 | 04/11/2013 | 15 |
| Estudo do MIL-STD-1553b | 05/11/2013 | 23/12/2013 | 35 |
| Mil-STD 1553b | 11/11/2013 | 17/01/2014 | 50 |

**Tabela 4 - Planeamento do primeiro semestre**

### 3.4.2. Segundo Semestre

A segunda fase do estágio constituiu na execução dos testes, na análise dos resultados e na afinação das especificações e dos testes. A afinação das especificações e dos testes acontece quando se verifica que afinal estes não cobrem integralmente todos os requisitos, ou devido a detalhes que às vezes só são percetíveis aquando da execução. O processo de execução, análise e correção é bastante demorado, uma vez que, quando existe alguma falha é necessário verificar a origem da mesma. Estas falhas podem ser do *software* (cujo os nossos testes que fazemos pretendem identificar e indiretamente eliminar), da *Software Validation Facility* (SVF) ou até mesmo do próprio teste que poderá estar mal especificado ou implementado.

Para além do GNC, SADM e APM, os módulos estudados e sobre os quais foram criados parte dos testes no primeiro semestre (No Apêndice A podemos encontrar o Gantt do projeto com os nome dos testes criados e implementados no primeiro semestre), foram também executados alguns dos testes do TR e EDM. Estes últimos dois módulos são mais simples, e os seus testes foram especificados por outros membros da equipa de validação.

O restante tempo foi dedicado inteiramente à escrita da dissertação final, aprofundando mais os capítulos já abordados na entrega intermédia, e desenvolvendo novos capítulos, como por exemplo a análise dos resultados e o estudo científico sobre validação do *software*.

Na tabela seguinte estão representadas as tarefas a realizar planeadas no segundo semestre, assim como as respetivas datas de início, fim e a duração em dias úteis.

| | Início | Fim | Duração(dias úteis) |
|---|---|---|---|
| Execução dos testes do SADM | 20/01/2014 | 09/05/2014 | 80 |
| Execução dos testes do APM | 20/01/2014 | 09/05/2014 | 80 |
| Execução dos testes do GNC | 20/01/2014 | 09/05/2014 | 80 |
| Execução dos testes do TR | 15/05/2014 | 30/05/2014 | 15 |
| Execução dos testes do EDM | 15/05/2014 | 30/05/2014 | 15 |
| Relatório Final | 02/06/2014 | 01/07/2014 | 22 |

**Tabela 5 - Planeamento do segundo semestre**

# Capítulo 4
# *ExoMars Trace Gas Orbiter* (TGO)

Neste capítulo é descrito o funcionamento do *ExoMars Trace Gas Orbiter* (TGO). Para facilitar a compreensão do seu funcionamento, este capítulo irá começar com um subcapítulo onde é feita uma introdução aos sistemas aeroespaciais e de seguida será descrito o funcionamento do *ExoMars TGO*.

## 4.1. Sistemas Aeroespaciais

Os sistemas aeroespaciais são sistemas bastante complexos. Pois, os equipamentos são sujeitos a condições severas de operação, tais como diferenças de pressão, temperaturas acentuadas e elevadas cargas aplicadas a pontos estruturais críticos.

De forma a facilitar a compreensão do funcionamento do *ExoMars TGO*, será introduzida uma breve descrição da arquitetura genérica de um satélite, dos meios de comunicação (Telecomandos e Telemetria), a normalização das comunicações (*Packet Utilisation Standard*) e o padrão MIL-STD-1553.

### 4.1.1. Arquitectura

A grande generalidade dos satélites é bastante idêntica contendo, como por exemplo, a estrutura principal/base, os painéis solares, bateria, controle térmico, propulsores para controle da *attitude*[1], descodificadores e codificadores e respetivas antenas para comunicação com a terra, etc.

A carga do satélite diz respeito aos componentes específicos que servem diretamente o propósito para o qual a missão foi criada, e que como tal é diferente para cada missão espacial. No caso do *ExoMars TGO* temos os componentes para medição dos gases da atmosfera marciana, o módulo para estudo da entrada de marte, e o sistema de comunicação UHF para suportar/reencaminhar a comunicação entre o módulo de entrada e a terra.

Uma vez separado do lançador que o deixa numa determinada orbita, este tem desde logo que utilizar os seus equipamentos de localização (como o *star tracker*) e deteção de inercia para se localizar com precisão e verificar a sua *attitude*[1]/inércia.

Posto isto, terá um tempo limitado para colocar-se na orbita ou trajetória a seguir e procurar a direção do sol, para abrir e orientar os seus painéis solares e assim não esgotar as suas baterias.

As correções às trajetórias/orientação/orbitas/*attitudes*[1] são normalmente feitas por libertação de substâncias gasosas a alta pressão num determinado ponto da estrutura do satélite (uso limitado) ou por intermédio de rodas de grande inércia a rodar a grandes velocidades e em diferentes eixos (variar de forma controlada a rotação de alguma dessas rodas de inércia, leva um movimento da estrutura sobre si mesma).

[1] *Attitude* é um termo usado para descrever o comportamento do satélite, mais concretamente a orientação para um dado objeto (normalmente sol ou terra).

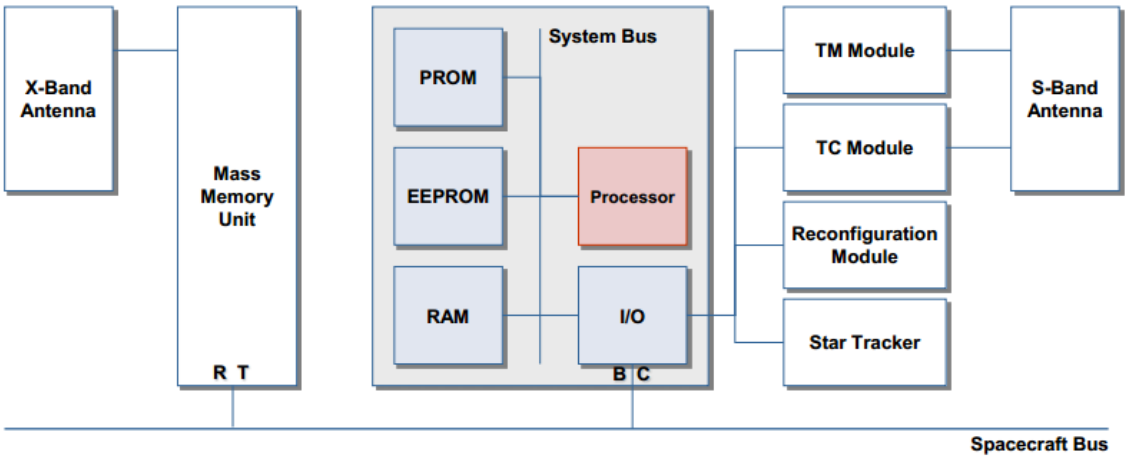



**Figura 11 - Arquitetura do computador de bordo uma nave aeroespacial [14]**

Na Figura 11 está representada uma arquitetura de um computador de bordo de uma nave aeroespacial e algumas das unidades funcionais. Como é visível na figura, o computador de bordo é constituído [14] por:

- PROM – memória que contém o *software bootstrap*;
- EEPROM – memória onde está guardada a aplicação do computador central. Esta memória pode ser reescrita;
- RAM – memória de execução;
- Processador – unidade de processamento;
- I/O – unidade responsável pelo controlo de *inputs* e outputs dos diversos barramentos da nave (RS422, IEEE1355, Mil-STD 1553b);

Para além do computador de bordo podemos encontrar as seguintes unidades funcionais:

- Memória massa – unidade de memória onde se guarda a informação gerada quando a nave não está em linha de vista com a estação de receção na terra (caso de satélites em orbita não geoestacionária). Esta informação irá ser enviada, quando houverem condições que assim o permitam.
- Antena X-Band – Antena que utiliza frequências aproximadamente entre 7.0 a 11.2 GHz e permite o envio de uma maior quantidade de informação em comparação com a antena S-Band;
- Antena S-Band – Antena que utiliza frequências que podem variar entre os 2 e os 4 GHz;
- Módulo TM – Módulo responsável pela memória e envio de telemetria através da antena S-Band;
- Módulo TC – Módulo responsável pela memória e receção de telecomandos através da antena S-Band;
- Módulo de reconfiguração – Módulo responsável por operações de baixo nível de reconfiguração e recuperação;
- *Star tracker* – dispositivo ótico com elevada sensibilidade que mede a posição das estrelas usando fotocélulas ou uma câmara com o propósito de determinar com rigor a posição atual do satélite.

### 4.1.2. Telecomandos (TC)

Os telecomandos são do tipo *uplink* e permitem o fluxo de informação da estação terrestre para o satélite. Através dos telecomandos a estação consegue solicitar o envio de informações do satélite, comandar manualmente o satélite, etc.

Na Figura 12 está ilustrada em detalhe a estrutura dos pacotes dos Telecomandos de acordo com o *Packet Utilization Standard* (PUS)[7][15], um *standard* definido pela Agência Espacial Europeia (ESA) para normalizar a comunicação dos seus satélites.

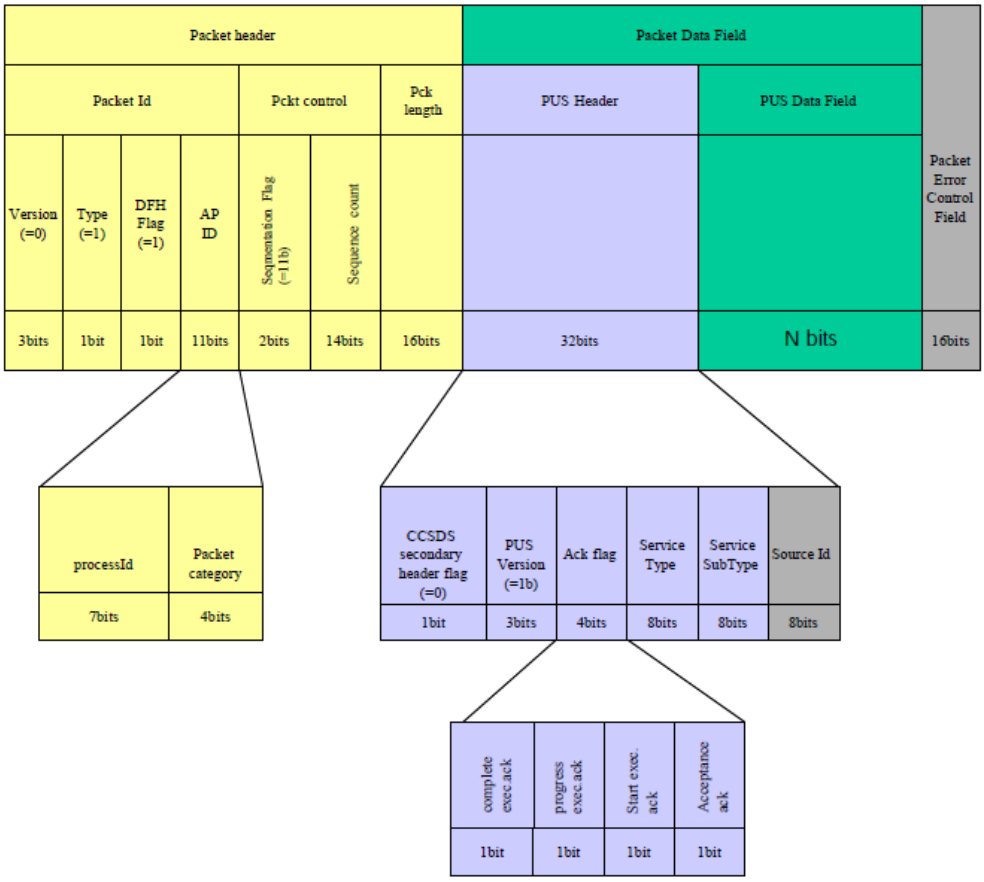



**Figura 12 - Estrutura de um pacote de telecomando [7][15]**

### 4.1.3. Telemetrias (TM)

As telemetrias são do tipo *downlink* e permitem o fluxo de informação do satélite para a estação de controlo em terra. Através das telemetrias a estação consegue saber informações do satélite como as temperaturas, as pressões, as voltagens, o estado dos equipamentos, etc.

Na Figura 13 está ilustrada em detalhe a estrutura dos pacotes dos Telemetria. Esta estrutura foi utilizada algumas vezes no processo de execução. Um dos serviços de telemetria é reportar a aceitação ou rejeição dos telecomandos. Quando existe uma rejeição, uma das possíveis maneiras de verificar essa rejeição é analisando os parâmetros do pacote.

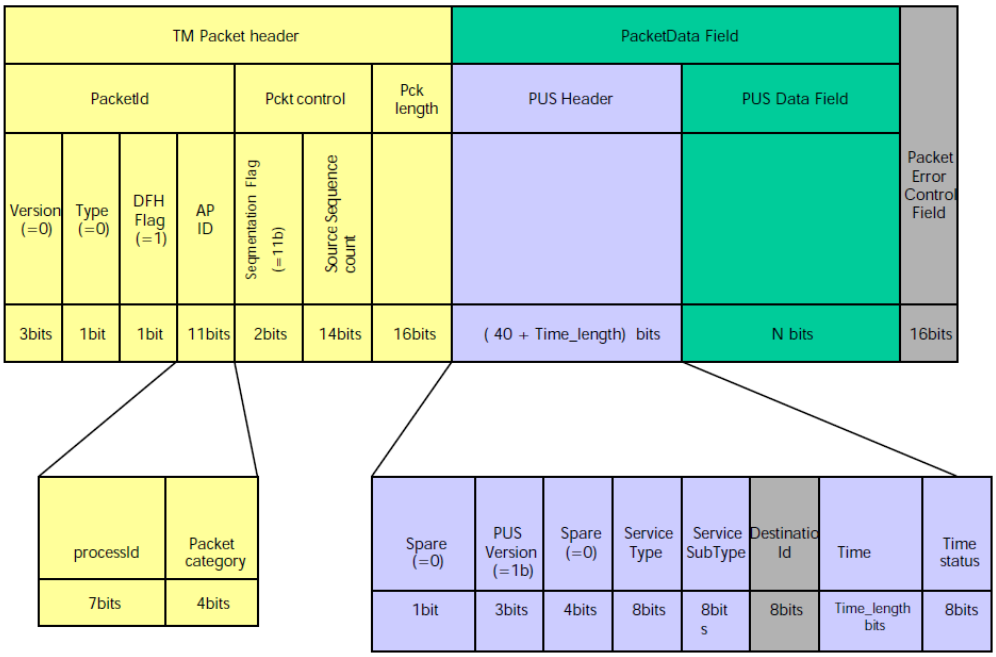



**Figura 13 - Estrutura de um pacote de telemetria[7][15]**

### 4.1.4. *Packet Utilisation Standard* (PUS)

O PUS é uma normalização das comunicações com um satélite e é definida pela *Consultative Committee for Space Data Systems*[7][15] (CCSDS).

Esta normalização baseia-se na subdivisão das comunicações por serviços e sub-serviços.

Estes serviços são numerados e a sua numeração vai de acordo como o seguinte critério:

- De 0 a 127 são reservados para os serviços genéricos. Nestes serviços os sub-serviços de 0 a 127 são reservados para os genéricos, de 128 a 255 são reservados para sub-serviços personalizados.
- De 128 a 199 são reservados para serviços personalizados que podem ser usados em várias missões.
- De 200 a 255 são reservados para serviços personalizados para uso numa missão específica.

Alguns dos serviços genéricos são [7][15]:

- SERVICE 1: TC VERIFICATION
- SERVICE 2: DEVICE COMMAND DISTRIBUTION
- SERVICE 3: HOUSEKEEPING REPORTING
- SERVICE 4: PARAMETER STATISTIC REPORTING SERVICE
- SERVICE 5: EVENTS REPOTING
- SERVICE 6: MEMORY MANAGEMENT
- SERVICE 8: FUNCTIONS MANAGEMENT
- SERVICE 9: TIMEMANAGEMENT
- SERVICE 11: ONBOARD OPERATION SCHEDULING MANAGEMENT
- SERVICE 12: ON-BOARD MONITORING
- SERVICE 13: LARGE DATA TRANSFER
- SERVICE 14: PACKET FORWARDING CONTROL SERVICE

- SERVICE 15: ON-BOARD STORAGE AND RETRIEVAL
- SERVICE 16: ON-BOARD TRAFFIC MANAGEMENT
- SERVICE 17: CONNECTION TEST
- SERVICE 18: OBCPMANAGEMENT
- SERVICE 19: EVENT / ACTION
- SERVICE 20: CONTEXT SAVING
- SERVICE 21: INFORMATION DISTRIBUTION
- SERVICE 22: SCIENCE DATATRANSFER
- SERVICE 128: MODE MANAGEMENTSERVICE

Alguns dos serviços específicos do *ExoMars TGO* são:

- SERVICE 129 : PARAMETER MANAGEMENTSERVICE
- SERVICE 132: ACTION SEQUENCE MANAGEMENT SERVICE
- SERVICE 134: UNIT CONFIGURATION MANAGEMENT SERVICE
- SERVICE 135: EQUIPMENT HARDWARE MANAGEMENT SERVICE
- SERVICE 136: PARAMETER EXTRACTION SERVICE
- SERVICE 142: FUNCIONAL MONITORING SERVICE
- SERVICE 144: FILE SYSTEM MANAGEMENT
- SERVICE 145: TC FILE MANAGEMENT
- SERVICE 150: TELEMETRY PACKET ROUTING
- SERVICE 170: PM APPLICATION SERVICE
- SERVICE 171: TC APPLICATION SERVICE
- SERVICE 172: TM APPLICATION SERVICE
- SERVICE 179: GNC MANAGEMENT SERVICE
- SERVICE 180: SADM MANAGEMENT SERVICE
- SERVICE 181: APM MANAGEMENT SERVICE
- SERVICE 183: TR MANAGEMENT SERVICE
- SERVICE 193: EDM MANAGEMENT SERVICE

Como referido, cada serviço contêm um ou mais sub-serviços no caso do serviço 128 (um dos mais usados no estágio) podemos encontrar os seguintes sub-serviços [7][15] s:

- TC(128,128) Set the function mode
- TC(128,129) Report the function mode
- TM(128,130) Function Mode report
- TC(128,131) Re-init Mode

### 4.1.5. Mil-STD 1553b

MIL-STD-1553 é um padrão de origem militar publicado em 1973 pelo Departamento de Defesa dos Estados Unidos [16], que define as características mecânicas, elétricas e funcionais de um barramento de dados transmitidos em sequência. Foi inicialmente desenhado para a aviação militar, mas atualmente é muito utilizado na comunicação dos vários componentes dos equipamentos especiais. Ao longo da sua existência, o 1553 foi implementado em milhares de aplicações como plataformas de petróleo, sistemas de controlo de metro e da *International Space Station* (ISS).

No início dos anos 90 o Departamento de Defesa dos Estados Unidos cedeu o 1553 à *Society of Automotive Engineers* (SAE). Esta continua a ser ainda hoje a responsável pelo *standard*.

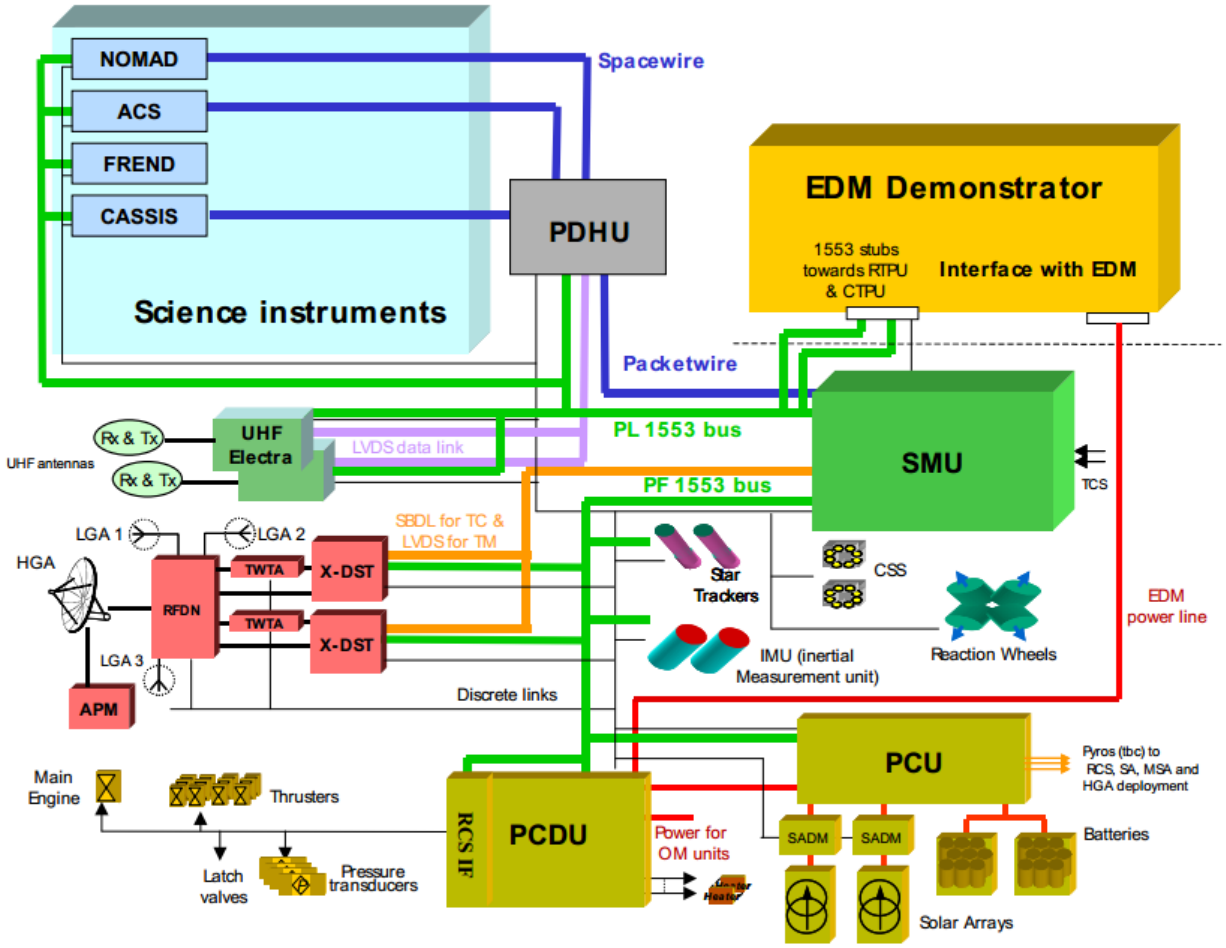



**Figura 14 - Comunicações no *ExoMars TGO*[3]**

Na Figura 14 estão representados os vários componentes do *ExoMars* TGO e as respetivas comunicações. Como se pode verificar na imagem, são utilizados dois barramentos MIL-STD-1553 [3]:

- O PL 1553 bus faz a comunicação entre o *Spacecraft Management Unit* (SMU) e o *payload* do satélite.
- O PF 1553 bus faz a comunicação entre o *Spacecraft Management Unit* (SMU) e a restante plataforma.

No Capítulo 6 está descrito seu funcionamento, assim como todo o trabalho realizado.

## 4.2. Funcionamento do *ExoMars TGO*

Como descrito no subcapítulo 1.2, o *ExoMars* é composto por vários componentes. Neste subcapítulo descreve-se apenas o funcionamento dos componentes que fazem parte deste estágio.

### 4.2.1. *Antenna Pointing Mechanism* (APM)

O APM é responsável por controlar a direção da *High Gain Antenna* (HGA) [2]. Sendo um mecanismo de dois eixos onde o intervalo de elevação pode variar entre [-26º;+155º] para X e o intervalo de azimute entre [-70º;+70º]. Em termos de estrutura o APM é dividido em duas camadas:

- Camada de função: suporta todos os algoritmos necessários para as funções de processamento de dados, navegação, controle e comando;
- Camada de gestão de modo: responsável pelo fluxo de dados (TC, TM, sequenciamento das funções implementadas na camada de função) e do modo de comutação no componente APM.

O APM está dividido em três modos para cobrir as diferentes fases da missão:

- MANUAL: O modo manual é utilizado em circuito aberto para executar um comando manual. Sendo o primeiro modo do satélite quando este inicializa.
- SAFE: O modo de segurança é usado para as antenas retornarem a uma posição de segurança.
- CRUISE: O modo Cruise é usado para a navegação, é o modo nominal.

O componente APM segue a máquina de estado ilustrada na Figura 15.

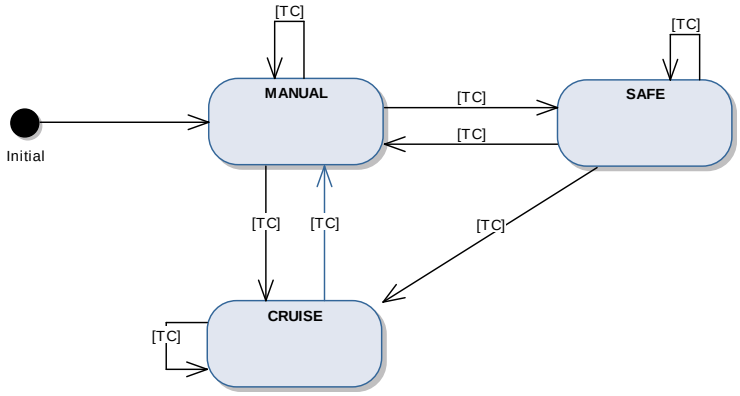



**Figura 15 - Máquina de estado do componente APM e SADM [2]**

As transições apresentadas na Figura 15 são executadas depois da receção de um TC(128,128). Na inicialização da aplicação, o modo MANUAL é ativado. Após o lançamento, é executada automaticamente a transição para CRUISE. A transição de CRUISE para SAFE é nominalmente realizada via o modo MANUAL.

Em caso de reconfiguração, é ordenada a transição para SAFE por uma sequência de ações. Depois, em SAFE, a *High Gain Antenna* (HGA) irá girar no sentido da sua posição retraída.

### 4.2.2. *Solar Array Drive Motor* (SADM)

O SADM [2] é bastante parecido com o APM, uma vez que também é controlado por dois motores, sendo o SADM responsável pelo controlo dos motores de cada um dos dois painéis solares do ExoMars TGO. Tal como o APM, o SADM é dividido em duas camadas (função e gestão de modo) e em três modos (MANUAL, SAFE e CRUISE).

### 4.2.3. *Guidance, Navigation and Control* (GNC)

O GNC [2] é responsável pelo controlo de *attitude* do satélite. Tal como os componentes anteriores é dividido em duas camadas (função e gestão de modo).

O GNC está dividido em seis modos de voo para cobrir as diferentes fases da missão:

- SBM (*Stand-By Mode*): Este modo permite monitorizar o comportamento do satélite durante a abertura dos painéis solares e a preparação do sistema de propulsão após a separação do lançador.
- SAM (*Sun Acquisition Mode*): É neste modo que é feita a aquisição de sol. Independentemente da sua *attitude*, o modo SAM tenta obter e manter os painéis em direção ao sol. Ele usa vários sensores para determinar posição e *attitude* e *Reaction Control Thruster* (RCT) para a atuação.
- NOMP (*Nominal Mode with Propulsion*): O objetivo do NOMP é guiar o satélite segundo três eixos usando os *star tracker* (STR), *gyrometers* (GYR) e *Reaction Control Thruster* (RCT). Este modo também é usado como parte de uma sequência de emergência em caso de anomalia.
- NOMR (*Nominal Mode with Reaction Wheels*): Com um objetivo idêntico ao NOMP, mas é baseado no estimador *Gyro-Stellar* e quatro *Reaction Wheels* em configuração assimétrica. É usado para guiar ao longo de grande parte da missão (incluindo fase de cientifica).
- OCM (*Orbit Control mode*): Este modo deve gerir todas as estratégias de controlo de órbita modificando as características orbitais das diferentes fases da missão ExoMars.
- AEBM (*Aero-Braking Mode*): Este modo é usado durante a aproximação à orbita de marte e é responsável pelo desaceleração do satélite.

Na Figura 16 estão representados os modos do GNC e as respetivas transições permitidas. Todas estas transições são executadas por um Telecomando com exceção a transição de SAM para NOMP que também podem ser executada de forma automática. As transições a amarelo (SBM para NOMP e SBM para OCM) só podem ser executadas em caso *fail-op*.

No Apêndice G é possível observar o diagrama da Figura 16 mais detalhado.

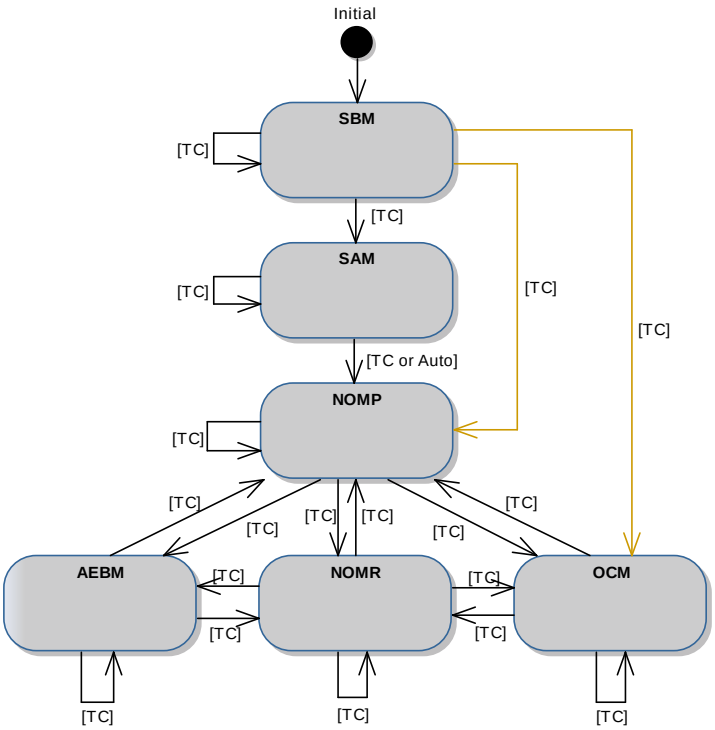



**Figura 16 - Máquina de estados do componente GNC [4]**

### 4.2.4. *Thermal Regulation* (TR)

O TR é um componente bastante simples e é responsável pela regulação da temperatura do satélite. Uma vez que os componentes têm uma temperatura mínima e máxima de funcionamento, é necessário aquecer os vários componentes para garantir o correto funcionamento.

### 4.2.5. Entry Descendent Module (EDM)

O EDM está ligado ao *Orbiter Module Bus* durante as primeiras fases da missão. Durante essas fases, o TGO irá transportar e fornecer energia e outros serviços (por exemplo, controle térmico e de comunicação em terra) para o módulo. Estes serviços irão permitir manter a bateria do EDM carregada e em bom estado de vida.

# Capítulo 5
# Processo de Validação

Neste capítulo descreve-se as atividades e as tarefas para o processo de validação.

O processo de validação do *ExoMars* é dividido em três atividades principais:

- Especificação dos testes de validação (*Software Validation Testing Specifications*);
- Codificação dos testes de validação (*Software Validation Test Procedures*);
- Execução dos testes de validação (*Software Validation Test Execution*);

Cada uma destas atividades é dividida em várias tarefas e são executadas para cada componente do *software* a ser validado. Como as atividades são dependentes é necessário que todas as tarefas sejam totalmente executadas para o sucesso de cada atividade.

Como demonstra no diagrama da Figura 17, um teste vai ter vários estados ao longo do seu percurso. Inicialmente é necessário escrever o *Software Validation Testing Specifications* (SVTS) de cada componente, cada SVTS vai ser composto por um ou mais testes e cada teste por um ou mais casos de teste. Ao escrever as SVTS são criados os testes. Para cada teste, se for possível validar pelo atual método de validação é criado o teste com o estado a *PLANNED*. Caso contrário o estado passa a *Out Of Scope* (OOS).

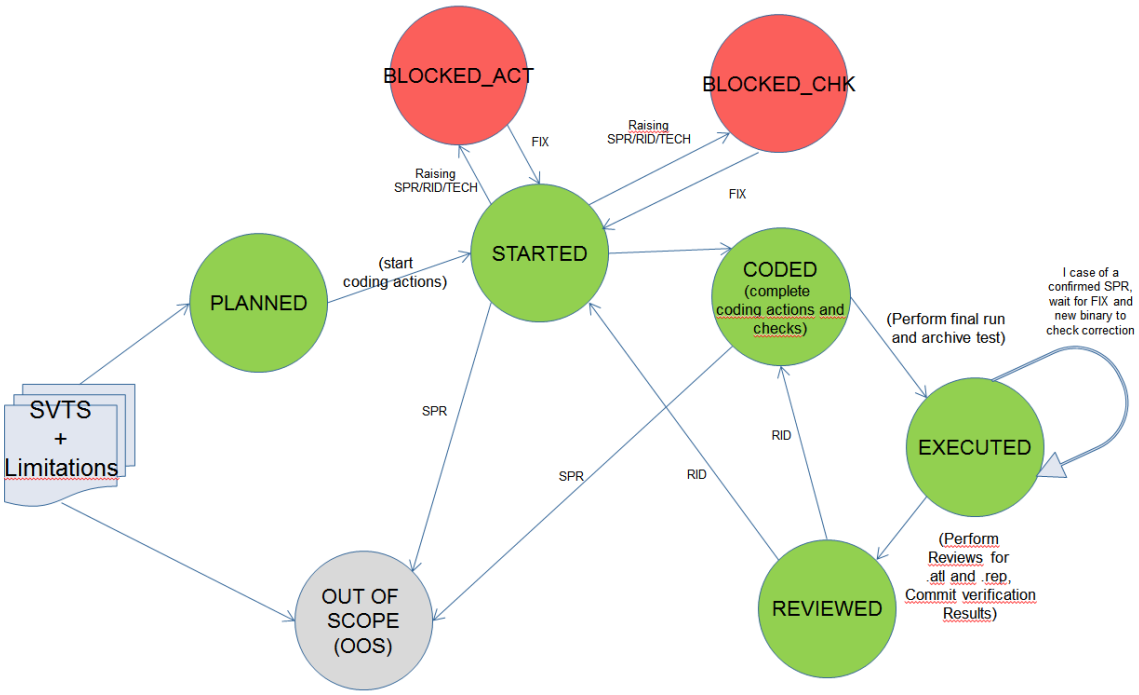



**Figura 17 - Ciclo de vida de um teste [13]**

Ao começar a implementação de um teste o seu estado passa para *STARTED*. A implementação começa pelas ações, ou seja, pelo envio de telecomandos no formato do *Packet Utilization Standard* (PUS) para o satélite, para este executar as ações desejadas.

Durante a implementação das ações pode ser encontrado algum problema (como por exemplo um erro no *Software Requirements Specification* (SRS), ou na base de dados, ou até mesmo nos serviços da SVF) sendo necessário reportar esse problema, passando o estado desse teste para *BLOCKED_ACT*. Quando todas as ações estão corretamente implementadas, começa a implementação das verificações, sendo o processo muito idêntico

ao das ações, mas quando é encontrado algum problema o estado altera-se, passando para *BLOCKED_CHK*. Quando as ações e as verificações estão todas implementadas, o estado altera-se para CODED.

A próxima fase é a execução, onde o teste é enviado para a *Software Validation Testing Specifications* (SVF), obtendo-se um relatório com o resultado de cada uma das verificações do teste. Caso todas as verificações estejam bem codificadas, o estado do teste passa para *EXECUTED*, independentemente se o teste falhou ou passou. Caso falhe deverá ser lançado(s) o(s) respetivo(s) *Software Problem Report*(s) (SPR). Caso o *Software Problem Report* seja confirmado pela equipa de desenvolvimento, é necessário esperar por um novo binário com a correção do problema, a fim de se fazer um nova execução do teste para aferir a correta correção.

## 5.1. Especificação dos testes de validação

Assim que exista uma versão estável do documento de especificação de requisitos de *software* para um determinado componente podemos iniciar esta primeira fase.

Dentro do componente tenta-se agrupar os requisitos por funcionalidade e criar testes para aferir cada grupo de requisitos encontrados.

Por vezes é difícil e desaconselhável verificar um requisito em apenas um teste, pelo que o mesmo poderá ser verificado parcialmente em vários testes.

Em outros casos (por exemplo requisitos relacionados com a inicialização de variáveis internas, não visíveis por telemetria) poderá ser difícil verificar determinado requisito por teste de sistema (são executados considerando apenas as interfaces externas do software, ou seja, com a mesma visibilidade e flexibilidade que tem o utilizador do sistema, neste caso os operadores do satélite). Nesses casos os requisitos podem ser integralmente ou parcialmente verificados por análises (geralmente inspeções de código).

Ao finalizar a criação de testes é necessário rever a rastreabilidade, para verificar que efetivamente todos os requisitos estão cobertos por testes e que quando existem coberturas parciais a determinados requisitos estas parcialidades se complementam de forma a verificar o requisito na íntegra.

Assim em suma, a especificação dos testes de validação é dividida nas seguintes tarefas:

- Escrever as especificaçoes de teste de validaçao para o componente;
- Realizar uma análise de rastreabilidade, para gerar uma matriz de rastreabilidade entre os requisitos e respetivas SVTS;
- Reportar problemas encontrados;

De seguida será detalhada cada uma destas subactividades.

### 5.1.1. Escrever as *Software Validation Testing Specifications* (SVTS)

Cada documento de especificação de testes de validação cobre as funcionalidades de um componente descrito no documento de especificação de requisitos (SRS, abreviado em inglês). Esta abordagem permite uma boa flexibilidade entre a implementação e a validação, uma vez que permite começar a STVS logo que a especificação técnica do componente esteja disponível. Cada SVTS irá cobrir completamente a especificação técnica de um componente por análise ou testes de validação. Irá também descrever quais dos requisitos do

SRS são cobertos por cada teste ou análise. Em alguns casos um SVTS pode cobrir parcialmente um requisito do SRS.

Quando um requisito da especificação técnica é proposto para ser validado por um teste, deve-se ter em consideração as funcionalidades disponíveis na SVF, o manual de utilização do *software* (SUM, abreviado em inglês) e a especificação técnica, para confirmar a possibilidade do requisito poder ser validado por teste. Caso contrário poderá ser validado por análise com a devida justificação, no entanto este método deve ser evitado em detrimento do primeiro, já que mais tarde os testes vão ser executados em equipamentos reais.

Cada especificação técnica cobre um ou mais componentes do *software* e cada componente do software contém várias funcionalidades.

Cada componente é coberto por uma especificação de testes de validação (SVTS, abreviado em inglês) que proporciona um bom nível de controlo sobre as atividades de validação, e tem as seguintes vantagens:

- Caso alguma funcionalidade de um determinado componente seja alterada, apenas é necessário alterar o respetivo SVTS;
- No caso de ser necessário aumentar as funcionalidades de um componente, apenas é necessário atualizar o SVTS desse componente. Evitando assim tocar em SVTS de outros componentes;

A estratégia de validação para cobrir os requisitos de especificação técnica para cada componente toma forma no respetivo SVTS, quando analisamos as características dos componentes e os agrupámos de acordo com diversas diretrizes, como a maximização do número de testes totalmente automáticos e como a separação de testes com execução complexa de testes com as execuções simples.

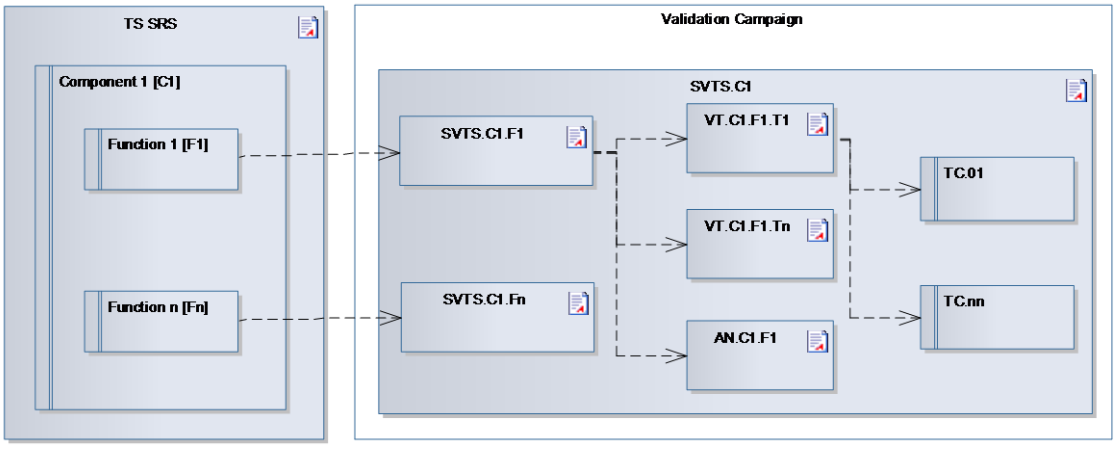



**Figura 18 - Exemplificação da validação de um componente [13]**

A Figura 18 exemplifica a SVTF para a funcionalidade F1 do componente C1, ou seja, SVTS.C1.F1 inclui:

- N testes VT.C1.F1.T[1..n], onde cada teste cobre características especificas do F1, como por exemplo inicialização, condições, transições, etc;
- Uma análise AN.C1.F1, para cobrir todos os requisitos de especificações técnicas não testáveis.

O componente C1 é dividido em várias funcionalidades F[1..n]. A SVTS.C1 contém os testes de validaçãoVT.C1.F[1..n] as várias funcionalidades do componente C1. Os testes

podem ser divididos em um ou mais casos de teste, no caso do VT.C1.F1.T1 é dividido em vários casos de teste TC.[01..nn].

Os SVTS são criados no *Enterprise Architect* (EA) usando a notação UML. O *Enterprise Architect* (EA) tem a grande particularidade de poder gerar essa informação em vários formatos diferentes e sobre vários Templates. No Apêndice B pode-se ver um exemplo de SVTS gerado para Word segundo os Templates definidos pela Critical Software.

### 5.1.2. Análise de rastreabilidade

Esta tarefa tem como *input* a especificação técnica de um componente e a respetiva SVTS, que contém os requisitos a serem cobertos em cada teste de validação ou análise.

Com base nas SVTS e documentação SRS a rastreabilidade entre os requisitos do SRS e a SVTS será gerada automaticamente. É esperada a cobertura total, e qualquer desvio deve ser devidamente justificado e registrado no respetivo SVTS.

Em caso de cobertura parcial de um requisito, esta informação deve ser mencionada na especificação do teste de validação.

Devido à possível presença de uma cobertura parcial de alguns requisitos, mesmo sendo a matriz de rastreabilidade gerada automaticamente, é necessária uma análise manual para garantir uma cobertura total, ou seja, que todas as parcialidades de cobertura de um requisito se complementam para atingir a cobertura total.

Todos os requisitos que não possam ser validados por este método de validação devem ser claramente identificados e justificados na matriz de rastreabilidade, incluindo os casos em que o requisito é coberto por testes unitários e de integração.

### 5.1.3. Reportar problemas encontrados

Esta tarefa tem como principal objetivo reportar os problemas encontrados durante a criação dos SVTS. Podem existir os seguintes tipos de problemas:

- Problemas na especificação técnica dos requisitos;
- Problemas na *Satellite Reference Database*[2] (SRDB);

No ponto 5.3.3 é explicado como é reportado um problema encontrado.

## 5.2. Codificação dos testes de validação

Nesta atividade implementou-se cada teste de validação descrito nos SVTS, gerando um conjunto de ficheiros *Advanced Test Language* (atl) a serem executados posteriormente na SVF. No Apêndice D pode-se encontrar um exemplo de um teste implementado.

A codificação dos testes de validação pode começar assim que o SVTS de um componente estiver completa.

A implementação de um teste do SVTS é dividida em duas partes: ações e verificações. Quando todas as ações e verificações forem implementadas, o teste é compilado numa linguagem específica da SVF, a *Advanced Test Language* (atl), que á baseado na linguagem de

[2] Base de dados com os valores mnemónica de cada parâmetro possível nos Telecomandos e Telemetrias.

programação C e que adicionalmente disponibiliza librarias de teste e comandos adicionais para uso da SVF.

Cada procedimento de teste cobre a respetiva especificação descrita no SVTS do componente. Excecionalmente, poderá existir mais que um procedimento para cobrir uma especificação de teste, como é exemplificado na Figura 19 onde na especificação de teste VT.C1.F1.T1 o caso teste TC.02 e o TC.03 são cobertos pelo VTP.02. Tal exceção decorre de limitações no número de ações/verificações possíveis por ficheiro atl, onde os testes são codificados.

A tarefa de codificação dos testes de validação inclui a implementação de código na linguagem atl garantindo que estes estejam em conformidade com as respetivas especificações do teste.

Cada procedimento deverá ter em consideração as restrições da SVF, ou seja, minimizar a complexidade de configuração e tamanho do teste, que pode afetar consideravelmente a complexidade da análise dos resultados e o tempo que cada teste leva para ser executado. A SVF será compartilhada entre toda a equipa de validação da Critical Software.

Por experiencia em projetos anteriores, a Critical Software considera preferível ter um maior número de casos de teste simples (mais atómicos) do que tentar cobrir muitas funcionalidades em simultâneo.

A codificação dos testes de validação é considerada completa para um determinado componente quando cada teste do seu SVTS está coberto pelos procedimentos e estes compilarem com sucesso.

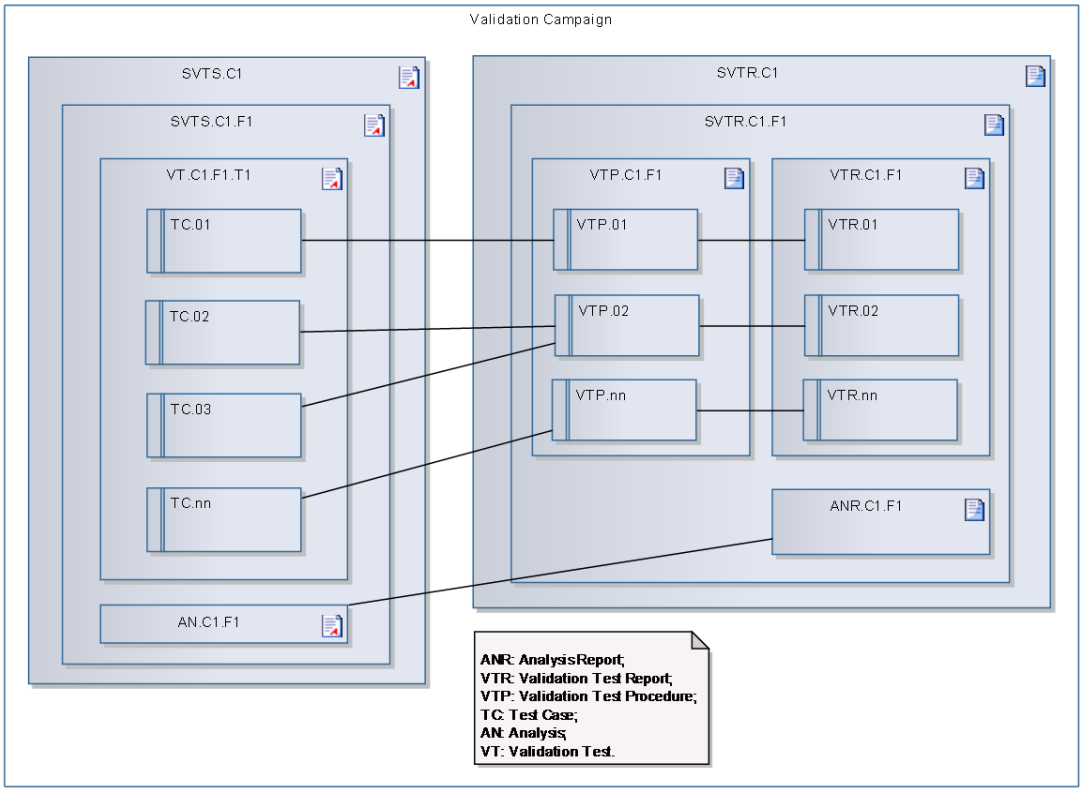



**Figura 19 - Exemplo incluindo uma SVTS e SVTR [13]**

## 5.3. Execução dos testes de validação

Nesta atividade executa-se cada teste de validação ou análise descrita nos Software Validation Testing Specifications (SVTS), gerando um conjunto de resultados.

A execução dos testes de validação é composta nas seguintes atividades:

- Execução dos testes, debug e recolha de resultados;
- Analise dos resultados da execução dos testes;
- Reportar problemas no *software*;
- Relatório de validação do *software*;

### 5.3.1. Execução dos testes, debug e recolha de resultados

Uma vez o teste compilado este é carregado para a SVF onde vai ser executado no simulador em tempo real. Quando o resultado do teste não corresponde ao comportamento esperado, o teste pode ser reajustado, recompilado e executado algumas vezes até que o comportamento esperado seja alcançado, isto caso as inconsistências resultem de problemas na especificação ou na implementação do teste.

A execução gere vários *outputs* que irão ser usados para análises de critérios PASS/FAIL. O processo de execução e geração de resultados é feito automaticamente pela SVF.

Quando algum resultado não for o esperado, a equipa de implementação pode ser consultada para esclarecimento de dúvidas sobre o componente.

Em caso de falhas identificadas no teste, estas são reportadas no JIRA para que a equipa responsável resolva o mesmo. Caso confirmado, é necessário esperar por uma nova versão do binário com essa correção para voltar a correr de novo o teste e verificar a correção.

### 5.3.2. Analise dos resultados da execução dos testes

A análise dos resultados dos testes consiste na verificação dos resultados da execução na SVF.

Depois da execução estar concluída, é necessário fazer a análise do teste. Para fazer a análise, basta executar um script que vai gerar o ficheiro *.rep. Este ficheiro de texto contem um cabeçalho que identifica o ambiente usado na execução, e depois para caso de teste são listados as suas ações, verificações e seus resultados e um sumário referindo o número total de ações, verificações manuais, verificações automáticas passadas e que falharam.

Caso seja necessário, e para investigação de problemas do teste poderão ainda ser gerados os seguintes outputs:

- O tráfego dos canais TC e TM (ficheiro *-tctm-traffic-1.csv);
- O tráfego da plataforma (ficheiro *-1553pf-traffic-1.csv) e *payload* (ficheiro *-1553pl-traffic-1.csv);
- O tráfego das linhas I/O (ficheiro *-smuio-traffic-1.csv);
- O tráfego do *SpaceWire* Bus (ficheiro *-spw-traffic-1.csv);

O mesmo pode ser visível na Figura 20.

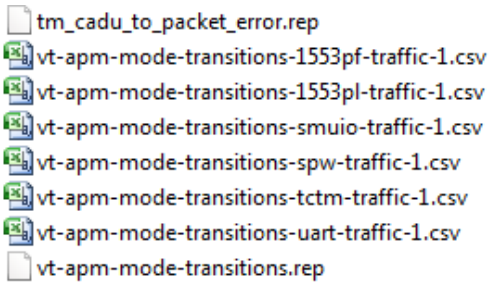



**Figura 20 - Output de execução de teste**

Relativamente à capacidade de verificações, a SVF fornece a execução de verificações automáticas nos seguintes outputs:

- O tráfego dos canais TM;
- O tráfego da plataforma e do *payload* (1553 Buses);
- O tráfego das linhas I/O;
- O tráfego do *SpaceWire Bus*;

O cumprimento dos requisitos que não podem ser validados pelos testes devido a restrições na SVF, por exemplo, a parte dos requisitos que não é observável nos resultados do teste de SVF (atribuições a variáveis não disponíveis para *downlink* por telemetria), deverá ser verificado por análises. A verificação por teste é sempre a preferível, pelo que cada análise deve ter registada no seu cabeçalho a justificação do seu uso em detrimento do teste funcional.

### 5.3.3. Reportar problemas no *software*

Sempre que é encontrado qualquer problema na validação (no *software*, SVF, etc), esse problema deve ser reportado. Antes de reportar o problema pode-se consultar a equipa de desenvolvimento quer a equipa da Thales Alenia Space que especificou os requisitos para esclarecimentos, de forma a garantir que existe mesmo um problema.

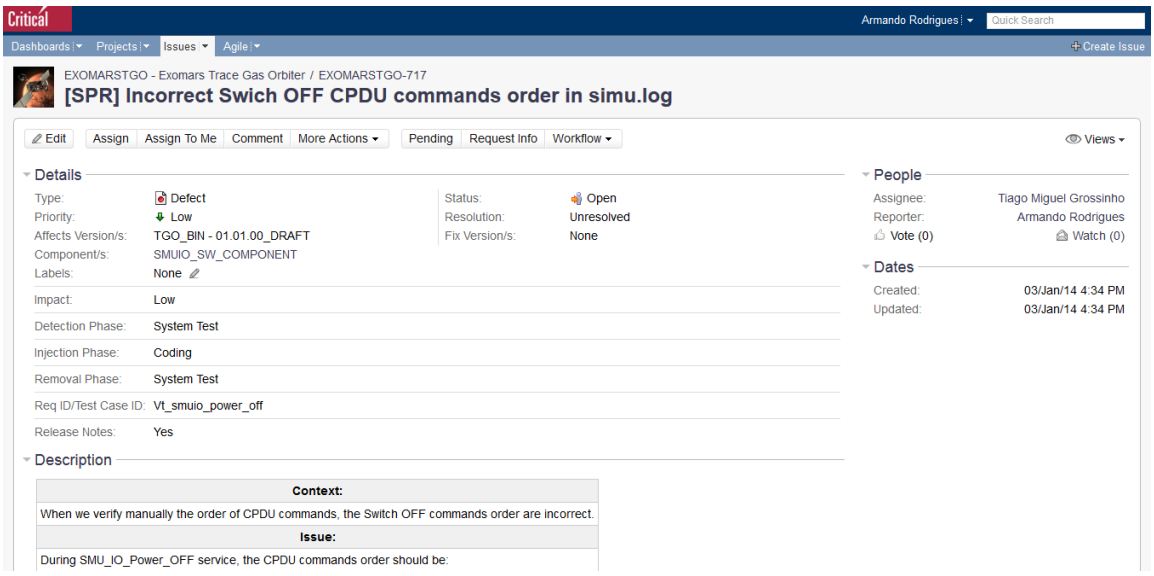



**Figura 21 - Issue criado no Jira**

Para reportar basta abrir a página do JIRA, descrever o problema e submeter o *issue*. Assim que a pessoa responsável por esse problema tenha disponibilidade, vai tentar resolver ou clarificar o problema e o JIRA notificará por *e-mail* a evolução do mesmo.

Na Figura 21 está um print de um *issue* criado durante o estágio.

### 5.3.4. Relatório de validação do *software*

Os *Software Validation Test Report* (SVTR) são gerados automaticamente quando os testes são executados. No Apêndice E está uma parte de um SVTR. No início destes relatórios podemos encontrar uma página de sumário. Essa página identifica:

- Descrição dos itens de configuração utilizados pela execução do teste, por exemplo, SDB, SVF, modelos de simulação de equipamentos digitais, etc;
- Quem executou o teste;
- Data da execução do teste;

Para além do sumário inicial, no início de cada caso de teste pode-se encontrar um sumário mais simples, onde indica:

- Qual o caso de teste;
- Número de casos manuais;
- Número de verificações automáticos passadas;
- Número de verificações automáticas falhados;

O SVTR também contém um sumário para cada análise, esse sumário identifica:

- Descrição dos itens de configuração utilizados pela análise, por exemplo, nome, versão, etc;
- Operador de análise;
- Data;
- O resultado (PASS/FAIL);
- O canal de comunicação utilizado;

# Capítulo 6
# MIL-STD-1553b

Como referido no planeamento, no início do estágio o projeto sofreu uma ligeira alteração. Devido à alteração inseriu-se no estágio a implementação de uma biblioteca na linguagem Java com o objetivo de ajudar a verificar, a construção e presença de mensagens no MIL-STD-1553b do *ExoMars TGO*.

Sem esta biblioteca, para ler ou escrever uma mensagem no canal do 1553 era necessário criar a mensagem bit a bit e enviar. Com a biblioteca o trabalho de validação sobre este protocolo é bastante mais fácil, rápido e simples. Outra razão é o facto de muitas das operações repetirem-se em vários testes, com a biblioteca basta chamar a função com a operação pretendida com os devidos parâmetros.

Este capítulo inicia com a descrição do funcionamento do MIL-STD-1553b e depois descreve-se a estrutura da biblioteca e a sua implementação.

## 6.1. Funcionamento

O MIL-STD-1553b é um protocolo de comunicção multiponto master-slave [17].

Existe um terminal que actuará com controlador do canal e é este que inicia todas as mensagens. Poderão existir até 31 equipamentos terminais. Existe mais um ederecamento disponivel nos 5 bits da mensagem que identificam o terminal a que se destina a mensagem, mas este está reservado para mensagens de broadcast.

Cada equipamento liga-se ao *bus* através de um canal físico nominal e redundante (tal permite que em caso de erro numa mensagem esta possa ser retrasmitida automaticamente no canal redundante com o minimo delay).

As mensages são constituidas por pacotes de 20 bits. O primeiro pacote é o comando, ao qual poderá ser acupolado até 32 pacotes de 16 bits de informação.

O primeiro pacote de comando é sempre construido e enviado pelo controlador do canal (no presente caso o sofware de bordo do satélite) e identifica a que equipamento terminal envolvido na comunicação com o controlador, o tipo (subaddress), o número subsequente de pacotes de informaçao e a direção (leitura ou escrita).

Caso se trate de um comando os respectivos pacotes de informação serão construidos e enviados pelo controlador de canal 1553, caso contrario, tratando-se de uma aquisição, será o terminal referido no grupo de comando que terá que construir e enviar os pacotes de informação da mensagem.

No fim de cada mensagem é enviada mais um grupo de 16 para sinalizar a auzencia/presença de erros na mensagem.

Na Figura 22 estão representados os diferentes tipos de palavras e os seus formatos.

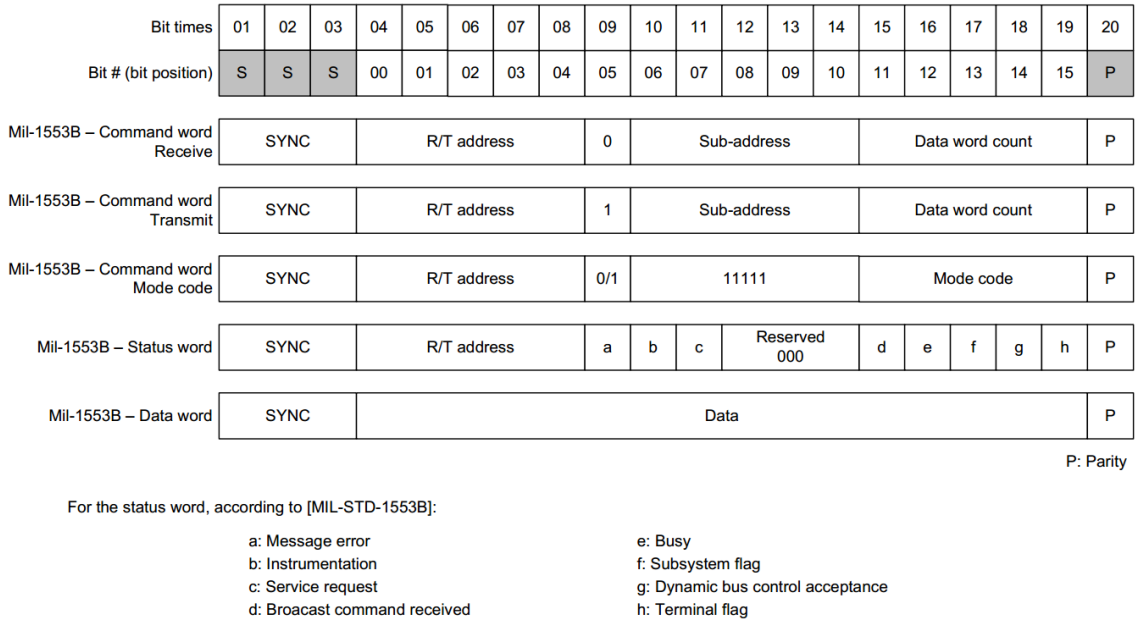



**Figura 22 - Tipos de mensagens no MIL-STD-1553b [18][19]**

Como podemos verificar na Figura 22, existem tês tipos diferentes de palavras [16] [18] [19]:

***Command Word*** - Especifica a função que um terminal remoto vai executar, pode executar funções de receção ou transmissão. É composta por *sync waveform*, *remote terminal address*, bit de transmissão / receção (1 para transmissão, 0 para receção), *Sub-address*, quantidade de palavras a enviar/receber e bit de paridade.

***Status Word*** - Usada para transmitir ao controlador se a mensagens foi bem recebida ou para transmitir o seu estado. É composta por *sync waveform*, *RT address*, bit de mensagem de erro, *instrumentation bit*, *service request bit*, três bits reservados, *broadcast command received bit*, *busy bit*, *subsystem flag bit*, *dynamic bus control acceptance bit*, *terminal flag bit*, e bit de paridade.

***Data Word*** - É nas *Data Words* que é transportada a informação das mensagens, tanto nas receções como transmissões. É composta por *sync waveform*, data bits, e bit de paridade.

## 6.2. Implementação

Para fazer as mensagens no canal 1553, temos que especificar diretamente todos os bits. Como a validação no módulo onde o 1553 integra é feita em Java, fazia sentido utilizar as propriedades *Object Oriented* da linguagem Java, de forma a representar cada tipo de mensagem que queremos verificar no bus.

Assim em vez de construir manualmente o pacote de uma determinada mensagem, a ideia foi criar e estanciar objetos de mensagens onde apenas é necessário especificar o terminal para qual a mensagem se dirige e a informação mais útil e específica de cada mensagem.

Posto isto, os objetos e o seu método check(), automaticamente construíam todo o *bitflow* da mensagem (de acordo com a especificação de cada mensagem) e chamavam a respetiva função de *check* do ambiente de validação.

No Apêndice F está ilustrado o diagrama de classes produzido através de *Reverse Engineering*. Como podemos verificar no diagrama, as mensagens são objectos B1553Message. Este objecto recebe como parametros o endereço e sub-endereço do terminal e o que vai escrever/ler, constroi toda a mensagem em binário e envia para o canal. Para construir as

*Command Word*, o objecto B1553Message utiliza a classe CmdWordGenerator, para as *Data Words* utiliza as classes WriteDataBlock e ReadDataBlock.

Num nível superior temos as classes MemoryRead e MemoryWrite que recebem um objeto do tipo B1553RT (objecto que contem o *Remote Terminal Address* da unidade a utilizar) e um Array com a informação a escrever ou ler, e vai construir as mensagens necessárias para enviar/ler toda a informação pretendida. Quando é escrita informação no canal, o método por defeito faz uma leitura para comparar a informação lida com a escrita. Quando se faz uma leitura, é feita uma comparação com a informação esperada.

As restantes classes são operações que foram usadas mais do que uma vez durante a validação do SMUIO. Desta forma, quando um teste necessitava de uma destas operações apenas foi necessário chamar o método correspondente.

# Capítulo 7
# Resultados

Nesta secção é apresentado um pequeno resumo de todo o trabalho realizado pelo estagiário ao longo do projeto de estágio *ExoMars*.

Como referido no capítulo anterior, durante o período de estágio foi desenvolvido uma biblioteca em Java para o MIL-STD-1553b.

O restante tempo foi dedicado à validação do *software* do *ExoMars TGO*, onde foram especificados, implementados e executados os seguintes testes dos seguintes componentes:

*Solar Array Drive Motor* (SADM):

- SADM TeleCommand Checks
- SADM Mode Transitions

*Antenna Pointing Mechanism* (APM):

- APM TeleCommand Checks
- APM Mode Transitions

*Guidance, Navigation and Control* (GNC):

- NOMP Submodes
- NOMR Unloading Submode
- OCM Submodes
- SAM Submodes
- Invalid Submode Transitions
- GNC TeleCommand Checks
- NOMR Submodes
- AEBM Submodes
- NOMP Slew Submode

A figura seguinte (Figura 23) ilustra os defeitos encontrados pela equipa de validação.

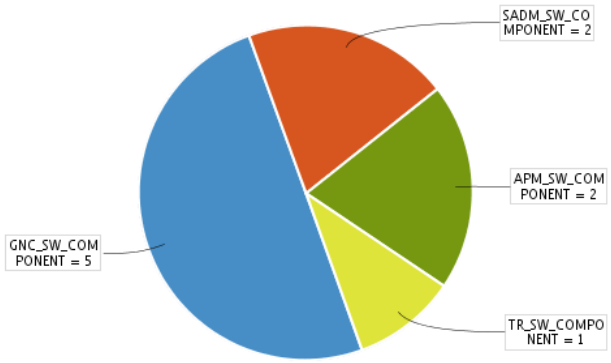



**Figura 23 - Defeitos por componente**

As figuras seguintes (Figura 24, Figura 25 e Figura 26) retiradas das estatísticas do Jira, ilustram os defeitos encontrados pela equipa de desenvolvimento.

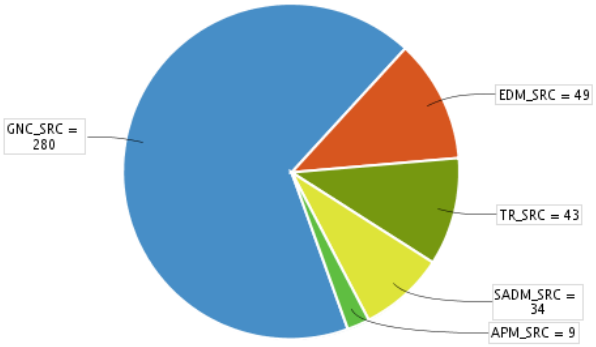



**Figura 24 - Defeitos encontrados por *Code Review***

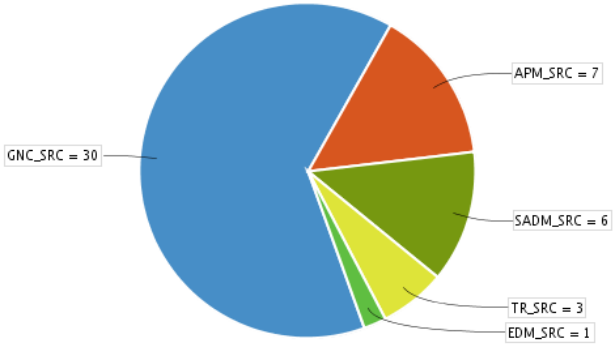



**Figura 25 - Defeitos encontrados por testes de unidade**

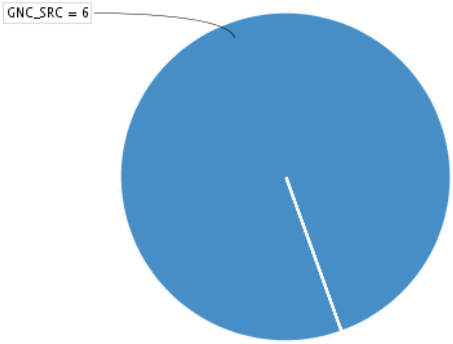



**Figura 26 - Defeitos encontrados por testes de integração**

Com base nos resultados ilustrados nas quatro figuras anteriores, podemos concluir que ao longo da evolução do projeto foram encontrados vários defeitos. Apesar de todas as verificações e testes da equipa de desenvolvimento, na validação do *software* foram encontrados defeitos. Como se trata de um satélite, após o lançamento é pouco provável conseguir corrigir um defeito, ou seja, se existir um defeito o mais provável é perder por completo o satélite e os milhões de euros de investimento. Em suma, a validação do *software* foi uma parte essencial do processo de desenvolvimento.

# Capítulo 8
# Conclusão

O presente projeto de estágio teve como principal objetivo a validação do *software* de bordo do *ExoMars Trace Gas Orbiter (TGO)*. Pode-se dizer que o objetivo proposto inicialmente foi cumprido, e até ultrapassado. Para além da validação prevista inicialmente, foi ainda desenvolvido uma biblioteca em Java para o MIL-STD-1553b.

O *ExoMars Trace Gas Orbiter (TGO)* é um satélite com destino a Marte com o propósito de estudar a sua atmosfera e testar a entrada, descida e aterragem de um módulo na sua superfície. O MIL-STD-1553 é um padrão de origem militar que define as características mecânicas, elétricas e funcionais de um barramento de dados serial.

Ao longo deste estágio, adquiri algum conhecimento extra, já que este projeto consiste numa área específica e rigorosa, uma vez que depois do seu lançamento não existe a possibilidade de correções. Este projeto requer conhecimento em Engenharia Aeroespacial. Por outro lado, salientar que durante a formação académica adquire-se pouco conhecimento sobre validação e/ou testes de *software*.

Durante o período de estágio, existiram vários aspetos a realçar. O projeto foi aliciante e desafiador, pois estive envolvido numa equipa experiente, multidisciplinar e com conhecimentos sólidos na área. Fui bem integrado na equipa e no projeto, tendo sempre todo o apoio necessário.

A passagem de conhecimento e de experiência profissional por parte dos orientadores deste estágio: Prof. Tiago Baptista, por parte da Universidade de Coimbra e Eng. Xavier Ferreira, da parte da Critical Software S.A., foi fundamental para o sucesso do projeto.

Como competencias adquiridas, é de realçar a autonomia na validação de *software*, ou seja, especificação, codificação, execução e análise de testes de forma independente.

Com a conclusão desta fase do projeto *ExoMars*, a fase seguinte vai consistir na realização de testes com equipamentos reais nas instalações da Thales.

Em suma, o *ExoMars* revelou-se uma experiencia única e enriquecedora ao nível pessoal, académico e profissional. Já que se trata de um projeto muito rigoroso da Agencia Espacial Europeia.

# Bibliografia

[1] Thales Alenia Space, "Glossary and Acronyms ExoMars Trace Gas Orbiter," *EXM-OM-LIS-CRT-0007-GLS*.

[2] Critical Software S.A., "Volume 2: Technical Proposal: ExoMars TGO Central Software Development," *CSW-2013-PRL-01097-vol2-exmtgo-csw-technical-proposal*.

[3] Thales Alenia Space, "ORBITER MODULE BUS (TGO)," *EXM-OM-DRP-AF-0425-03_00-TGO-design-report*.

[4] Thales Alenia Space, "TGO CSW GNC AOMD Software Requirement Specification," *EXM-OM-SRD-AF-0881_06 TGO CSW GNC AOMD SRS-no-track-changes*.

[5] Thales Alenia Space, "TGO CSW SADM SAMD Software Requirement Specification," *EXM-OM-SRD-AF-1128_06 TGO CSW SADM SAMD SRS-no-track-changes*.

[6] Thales Alenia Space, "TGO CSW APM APMD Software Requirement Specification," *EXM-OM-SRD-AF-1223_05 TGO CSW APM APMD SRS-no-track-changes*.

[7] Thales Alenia Space, "SMU SGICD Vol3," *EXM-OM-ICD-AF-0879 iss 7 - SGICD vol3*.

[8] Thales Alenia Space, "ExoMars TGO Software: SMS PUS Interface Control Document," *exm-tgo-icd-sms-pus-v1-draft*.

[9] S. McConnell, Software Quality at Top Speed, Microsoft Press, 1996.

[10] J. M. &. D. Conway, "The Software Development Process," 2005. [Online]. Available: http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/07.13.SWEng1/html/text.html.

[11] Mitch Lacey & Associates, Inc, "Inc. Introduction to Agile Software Development," [Online]. Available: http://www.mitchlacey.com/intro-to-agile/scrum.

[12] Sommerville, Software Engineering, Addison-Wesley, 2000.

[13] Critical Software S.A., "SValP Diagrams," *exomars-svalp-diagrams*.

[14] Critical Software S.A., "Space Systems Engineering," *CSW-2008-PRS-06666-space-systems-engineering*.

[15] Thales Alenia Space, "SGICD Volume 2 Generic TM and TC Interface," *EXM-MS-ICD-AI-0036_Iss_07 - SGICD vol2*.

[16] United States Department of Defense, "MIL-STD-1553b," [Online]. Available: http://snebulos.mit.edu/projects/reference/MIL-STD/MIL-STD-1553B.pdf.

[17] "What is MIL-STD-1553," [Online]. Available: http://www.milstd1553.com/.

[18] Alta Data Technologies LLC , "MIL-STD-1553 Tutorial and Reference," [Online]. Available: http://www.altadt.com/support/tutorials/mil-std-1553-tutorial-and-

reference.

[19] Granite Island Group, "MIL-STD-1553 & 1773 DATA BUS," [Online]. Available: http://www.tscm.com/1553-bus.pdf.

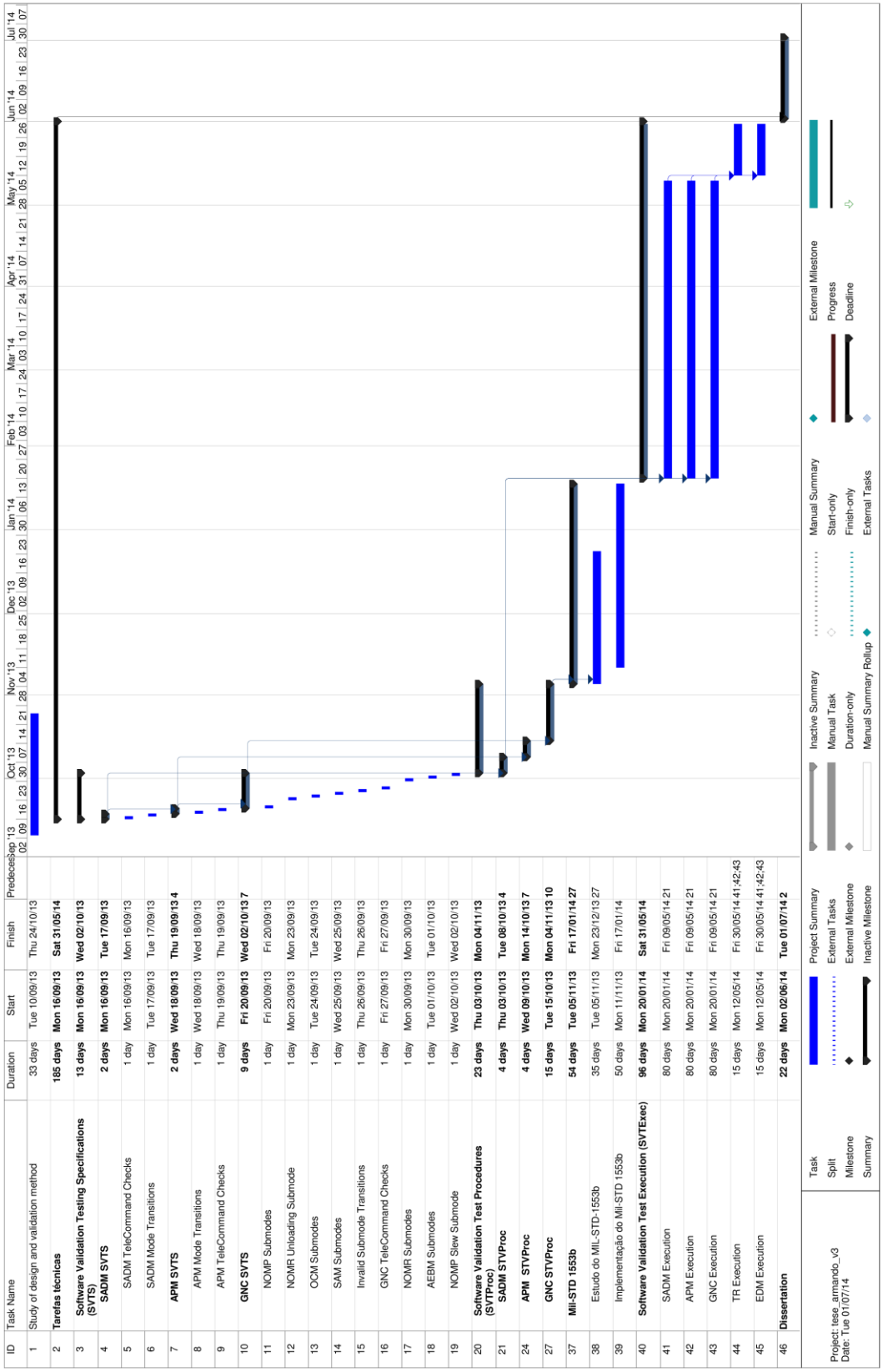
[20] Critical Software S.A., "Software Testing Process," *CSW-QMS-2004-SEP-2236-sw-testing-process*.

[21] Critical Software S.A., "Unit Testing Procedure," *CSW-QMS-2002-PRO-0513-unit-testing*.

[22] ESA, "THE EXOMARS PROGRAMME 2016-2018," [Online]. Available: http://exploration.esa.int/mars/46048-programme-overview/.

# I Apêndices

# Apêndice A
# Gantt do Projeto



# Apêndice B
# Exemplo de Requisito

O presente apêndice apresenta um dos requisitos do teste "APM Mode Transitions".

[EXM-TGO-CSW-RB-APMD-REQ-0045 a]

When an APM mode transition has been executed, a "Normal/Progress Report" TM(5,1) packet called RTM_AP_MODE_TRANSITION shall be sent with the previous mode and the new mode as parameters.

# Apêndice C
# Exemplo de SVTS

O presente apêndice apresenta a especificação para o requisito do Apêndice B.

Actions:

Send TC(128,128) to APM, providing the correct APM mode ID and requesting a transition to CRUISE.

Send TC(128,128) to APM, providing the correct APM mode ID and requesting a transition to SAFE.

Send TC(128,128) to APM, providing the correct APM mode ID and requesting a transition to MANUAL.

Checks:

TM(5,1)

Verify that three TM(5,1)-RTM_AP_MODE_TRANSITION reporting APM mode transitions are received and that on them previous and new mode are correctly reported.

# Apêndice D
# Exemplo de SVTProc

O presente apêndice apresenta a codificação para a especificação do Apêndice C.

Apenas é apresentado as acções e cerificações.

```
/**********************/
/*      Actions       */
/**********************/
//Send TC(128,128) to APM, providing the correct APM mode ID and requesting a transition to CRUISE.
 Send_TC_OPS(TC_128_128_AP_MODIFY_MODE,  TCP_128_128_AP_MODE_ID, APM_MODE_ID, TCP_128_128_AP_MODE, APM_MD_CRUISE);
 TM_1_7++;
//Send TC(128,128) to APM, providing the correct APM mode ID and requesting a transition to SAFE.
 Send_TC_OPS(TC_128_128_AP_MODIFY_MODE,  TCP_128_128_AP_MODE_ID, APM_MODE_ID, TCP_128_128_AP_MODE, APM_MD_SAFE);
 TM_1_7++;
//Send TC(128,128) to APM, providing the correct APM mode ID and requesting a transition to MANUAL.
 Send_TC_OPS(TC_128_128_AP_MODIFY_MODE,  TCP_128_128_AP_MODE_ID, APM_MODE_ID, TCP_128_128_AP_MODE, APM_MD_MANUAL);
 TM_1_7++;

/**********************/
/*      Checks        */
/**********************/
//TM(5,1)
//     Verify that three TM(5,1)-RTM_AP_MODE_TRANSITION reporting APM mode transitions are received and that on them previous and new mode are correctly reported.
//new mode
 Check_Parameter_VC_W_Msg("AP                                                    : RTM_AP_MODE_TRANSITION_NEW_MODE",
   TM_0, "Params([5]:[1])", "DAP0502Z", "SAME_ORDER", "AUTO", "CRUISE && SAFE && MANUAL");
//old mode
 Check_Parameter_VC_W_Msg("AP                                                    : RTM_AP_MODE_TRANSITION_OLD_MODE",
   TM_0, "Params([5]:[1])", "DAP0501Z", "SAME_ORDER", "AUTO", "MANUAL && CRUISE && SAFE");
```

# Apêndice E
# Relatório de execução de teste

O presente apêndice apresenta o relatório de execução do código do Apêndice C.

```
************************************************************************
* T E S T   R E P O R T
*=======================================================================
* Thales Space Industries
*=======================================================================
* Programme                          : exomars
* Test phase                         : VT
* Tested SW                          : OBSW
* Tested SW version/edition          : 02.00.00
* SDB version/edition                : arodrigues_sdb(TGO_02_02)
* SGSE version/edition               : EBVLEXM_04_00_01
*=======================================================================
* Execution test engineer            : arodrigues
* Execution begin date               : "2014-04-30 11-12-36"
* Execution end date                 : "2014-04-30 11-25-01"
* Execution status                   : Nominal
*=======================================================================
* Analyse checker mode               : On
* Analyse detail level               : Verbose
* Analyse date accuracy              : 1000us
* Analyse start date                 : 2014-04-30 11-58-39
*=======================================================================
* Processor module id                : PM A
* Simulation date                    : 2014-04-29 17-28-19
* Format TM                          : BVL4000
*=======================================================================
* Test                               : vt_apm_mode_transitions
* Test procedure version/edition     : Revision: 6491
************************************************************************


************************************************************************
* Test case 1010
************************************************************************

------------------------------------------------------------------------
- Automatic control : Check_1010_1
      Channel       : S-Band Telemetry : VCA_0
- Start date        : [01/01/2000 12:02:53.382360]
      Stop date     : [01/01/2000 12:04:56.431160]
- Result            : PASSED
- Check_Packet_Number(S_TM_VCA0,'Params([1]:[2])') == 0 ?
- Check the number of received TM_1_2
------------------------------------------------------------------------
Condition : == (0)
1010.116.1     Number of occurrences : 0
------------------------------------------------------------------------
- Manual control    : Print_1010_1
      Channel       : S-Band Telemetry : VCA_0
- Start date        : [01/01/2000 12:02:53.382360]
      Stop date     : [01/01/2000 12:04:56.431160]
- Print_Packet(S_TM_VCA0,'Params([1]:[2])')
- Debug Print - TM 1_2
------------------------------------------------------------------------
```

```
1010.117.1 No data match the request

------------------------------------------------------------------------
- Automatic control : Check_1010_2
      Channel       : S-Band Telemetry : VCA_0
- Start date        : [01/01/2000 12:02:53.382360]
      Stop date     : [01/01/2000 12:04:56.431160]
- Result            : PASSED
- Check_Packet_Number(S_TM_VCA0,'Params([1]:[4])') == 0 ?
- Check the number of received TM_1_4
------------------------------------------------------------------------
Condition : == (0)
1010.118.1     Number of occurrences : 0
------------------------------------------------------------------------
- Manual control    : Print_1010_2
      Channel       : S-Band Telemetry : VCA_0
- Start date        : [01/01/2000 12:02:53.382360]
      Stop date     : [01/01/2000 12:04:56.431160]
- Print_Packet(S_TM_VCA0,'Params([1]:[4])')
- Debug Print - TM 1_4
------------------------------------------------------------------------
1010.119.1 No data match the request
------------------------------------------------------------------------

[...]

------------------------------------------------------------------------
- Automatic control : Check_1010_6
      Channel       : S-Band Telemetry : VCA_0
- Start date        : [01/01/2000 12:02:53.382360]
      Stop date     : [01/01/2000 12:04:56.431160]
- Result            : PASSED
- Check_Parameter(S_TM_VCA0,'Params([5]:[1])',DAP0502Z) SAME_ORDER
CRUISE && SAFE && MANUAL && SAFE && CRUISE && MANUAL && CRUISE ?
- AP : RTM_AP_MODE_TRANSITION_NEW_MODE
------------------------------------------------------------------------
Condition : SAME_ORDER (CRUISE && SAFE && MANUAL && SAFE && CRUISE &&
MANUAL && CRUISE)
1010.134.1     [01/01/2000 12:04:05.760540]          DAP0502Z
0x02  CRUISE

1010.134.2     [01/01/2000 12:04:14.658830]          DAP0502Z
0x01  SAFE

1010.134.3     [01/01/2000 12:04:22.062130]          DAP0502Z
0x00  MANUAL

------------------------------------------------------------------------
- Automatic control : Check_1010_7
      Channel       : S-Band Telemetry : VCA_0
- Start date        : [01/01/2000 12:02:53.382360]
      Stop date     : [01/01/2000 12:04:56.431160]
- Result            : PASSED
- Check_Parameter(S_TM_VCA0,'Params([5]:[1])',DAP0501Z) SAME_ORDER
MANUAL && CRUISE && SAFE && MANUAL && SAFE && CRUISE && MANUAL ?
- AP : RTM_AP_MODE_TRANSITION_OLD_MODE
------------------------------------------------------------------------
Condition : SAME_ORDER (MANUAL && CRUISE && SAFE && MANUAL && SAFE &&
CRUISE && MANUAL)
1010.135.1     [01/01/2000 12:04:05.760540]          DAP0501Z
0x00  MANUAL
```

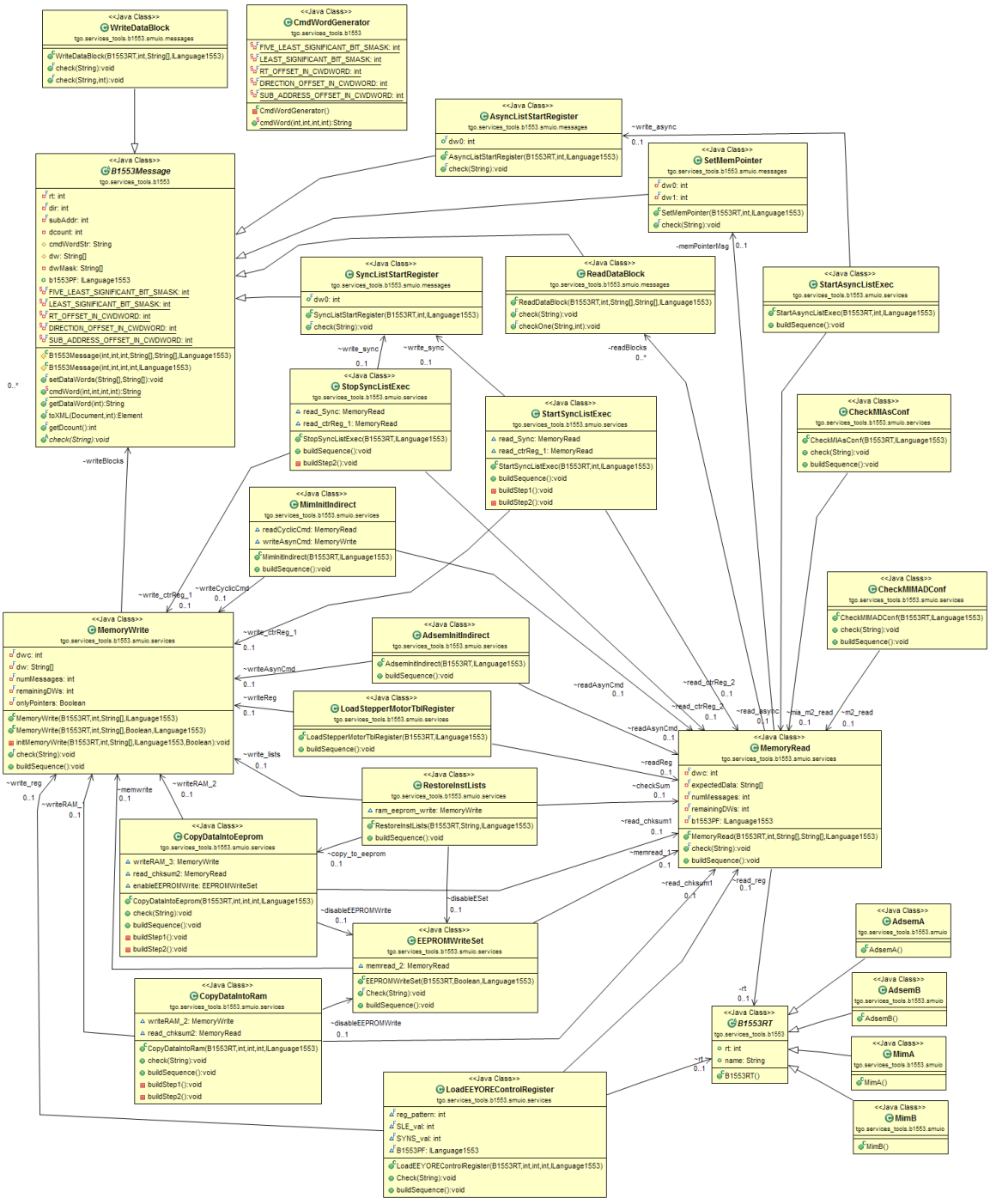
```
1010.135.2     [01/01/2000 12:04:14.658830]          DAP0501Z
0x02  CRUISE

1010.135.3     [01/01/2000 12:04:22.062130]           DAP0501Z
0x01  SAFE

*************************************************************************
* Test case 1010 conclusion
*=======================================================================
* Number of actions                  : 2
* Number of manual controls          : 0
* Number of automatic controls passed : 2
* Number of automatic controls failed : 0
*************************************************************************
```

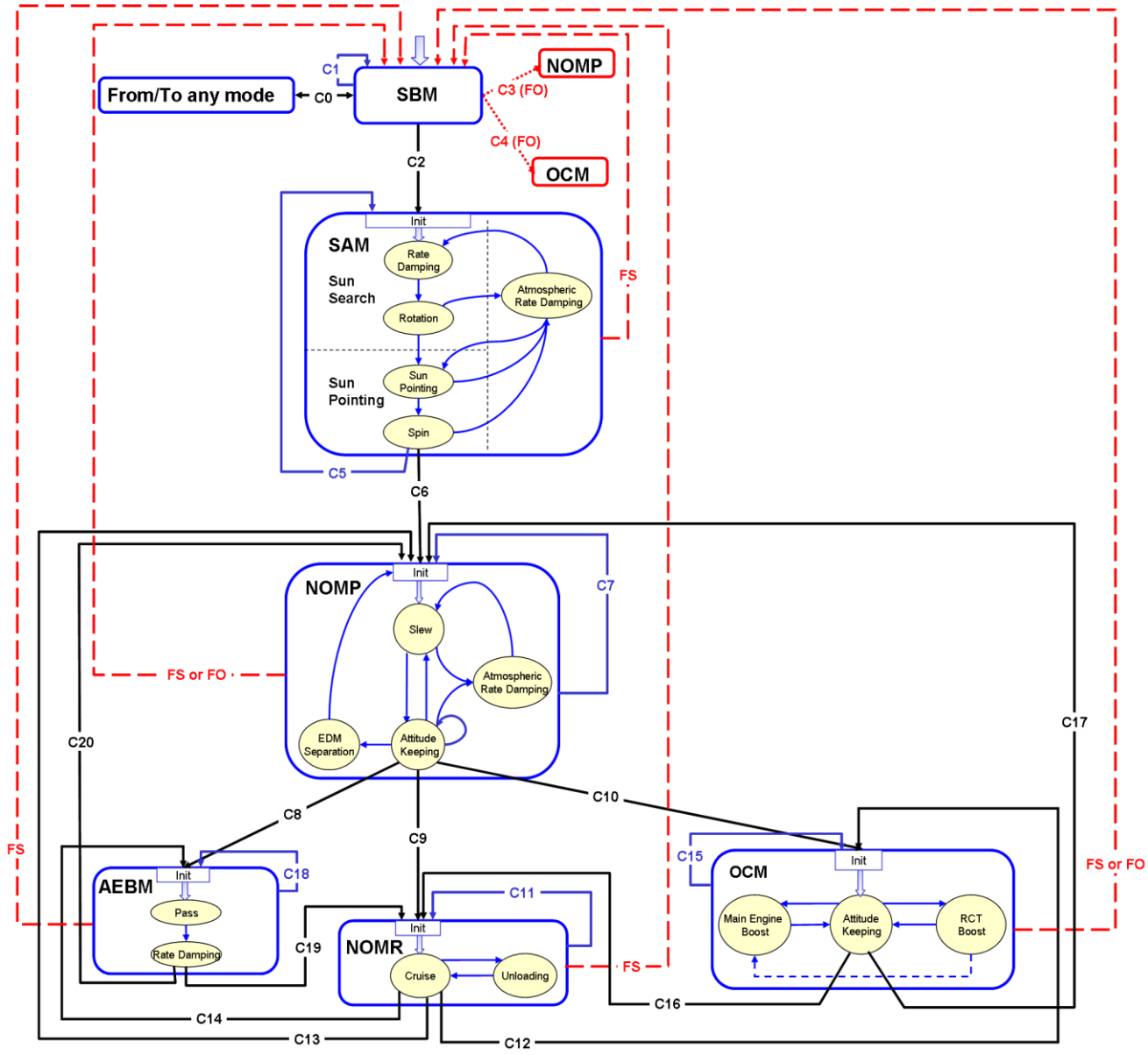
# Apêndice F
# Diagrama de classes da biblioteca para validação sobre o protocolo MIL-STD-1553b.

O presente apêndice ilustra a o diagrama de classes da biblioteca para auxiliar a validação sobre o protocolo MIL-STD-1553b.



# Apêndice G
# Máquina de estados do componente GNC

O presente apêndice ilustra a máquina de estados do componente *Guidance, Navigation and Control* (GNC) do *ExoMars TGO*[4].




As abreviaturas utilizadas na figura significam:

- "C" significa "Condition"
- "FS" significa "Fail Safe"
- "FO" significa "Fail Op"