SUPPLIER DOCUMENT Cover Page

idm@F4E UID / VERSION
2RBL9F / 1.2

VERSION CREATED ON / STATUS
04 January 2017 / Approved

EXTERNAL REFERENCE

Supplier Document

D4 MARTe Project Management Plan

Under the Specific Contract no. F4E-OFC-361-06, with subject "Fast plant controller prototype", this document is deliverable D4 MARTe Project Management Plan.

The scope of the current document is focused in the detailed description of the software development process for MARTe framework.

This document is intended for use by anyone interested in developing or expanding MARTe's ...

Approval Process			
	Name	*Action*	*Affiliation*
Author	**Herrero I.**	**04 January 2017:signed**	
Co-Authors			
Reviewers			
Approver	**Cabrita Neto A.**	**06-Jan-2017: approved**	**ITERD**
RO: Cabrita Neto Andre (F4E)			
Read Access	**LG: GTD team, LG: OFC-361-06-CCFE, AD: IDM_F4E, AD: F4E-A40_HEAD, AD: I-CODAC, AD: IDM IE-TS-CO-00 CODAC, GG: IAC, GG: IAS Audit on Document Management, project administrator, RO**		

Orig. Document MD5#: 6B51667AD4AF78F910735C6BA580ED2B

Change Log			
D4 MARTe Project Management Plan (2RBL9F)			
Version	***Latest Status***	***Issue Date***	***Description of Change***
v0.0	In Work	02 June 2015	
v1.0	Revision Required	02 June 2015	First version of the document.
v1.1	Approved	10 June 2015	Section 5.1.9: Typo correction in AD reference.
v1.2	Approved	04 January 2017	Coding unification with the other plans Add a clarification about destructors when unit testing.

Identification of the document			
Document Reference	01.ES-F4E-OFC-361-06 D4	Revision	1.2
F4E Reference	F4E_D_2RBL9F	F4E TRO	André C. Neto
F4E Customer Reference	N/A		
Date	2017-01-04		
Supplier	GTD SISTEMAS DE INFORMACION		
Graded Quality level	Class 3 – Any safety related item (SR) or non safety related item (NSR) whose failure could result in MODERATE impact.		

MARTe Project Management Plan

Fast plant controller prototype (F4E-OFC-361-06)

Summary

Under the Specific Contract no. F4E-OFC-361-06, with subject “Fast plant controller prototype”, this document is deliverable D4 MARTe Project Management Plan.

The scope of the current document is focused in the detailed description of the software development process for MARTe framework.

This document is intended for use by anyone interested in developing or expanding MARTe’s functionalities while harmonizing their contributions.

	Written by	Revised by	Approved by
Name	Ivan Herrero	Ivan Herrero	Javier Varas
Signature			
Date	2017-01-04	2017-01-04	2017-01-04

Contact person GTD: javier.varas@gtd.eu
Phone: +34 93 493 93 00

DISSEMINATION

Distributed to	Copies	Means
Project Team	1	e-mail
Fusion for Energy	1	Electronic

SUMMARY OF MODIFICATIONS

Edition	Date	Chapter	Modification	Author/s
1.0	02/06/2015	All	Document creation	IH
1.1	10/06/2015	5.1.9	Typo correction in AD reference.	HN
1.2	04/01/2017	Headers UT tasks	Coding unification with the other plans Add a clarification about destructors when unit testing.	IH

CONTENTS

CONTENTS **3**
0 Introduction **6**
0.1 Purpose and scope of the document 6
0.2 Relationship to other plans 6
0.3 Intellectual Property 6
0.4 Applicable and reference documents 6
0.5 Definitions, acronyms and abbreviations 7
1 Project Objectives and Constraints **8**
1.1 MARTe project objective 8
1.2 MARTe project scope 8
1.3 Technical description 8
2 Project Organization **9**
2.1 Organization 9
2.2 Communication & Contact 9
3 Configuration and Documentation **10**
3.1 Configuration Management 10
3.2 Documentation Management 10
3.2.1 Deliverables 10
4 Standards **12**
4.1 Software Coding Standard 12
5 Software activities **13**
5.1 Requirements activity 13
5.1.1 Requirements activity parameters 14
5.1.2 Requirements quality attributes 14
5.1.3 Requirements objectives 14
5.1.4 Requirements activity tasks 14
5.1.5 Tools used for requirements activity 15
5.1.6 Requirements activity transition criteria 15
5.1.7 Requirements' properties 15
5.1.8 Requirements activity closing action 16
5.1.9 Requirements activity verification 16
5.2 Architecture & design activity 16
5.2.1 Architecture & design activity parameters 16
5.2.2 Architecture & design quality attributes 16

5.2.3 Architecture & design objectives 16
5.2.4 Architecture & design activity tasks 17
5.2.5 Tools used for architecture & design activity 17
5.2.6 Architecture & design activity transition criteria 17
5.2.7 Architecture & design activity closing action 18
5.2.8 Architecture & design activity verification 18
5.3 Coding & documentation activity 18
5.3.1 Coding & documentation activity parameters 18
5.3.2 Coding & documentation quality attributes 18
5.3.3 Coding & documentation objectives 19
5.3.4 Coding & documentation activity tasks 19
5.3.5 Tools used for coding & documentation activity 19
5.3.6 Coding & documentation activity transition criteria 19
5.3.7 Coding & documentation activity closing action 20
5.3.8 Coding & documentation activity verification 20
5.4 Unit testing activity 20
5.4.1 Unit testing activity parameters 20
5.4.2 Unit testing quality attributes 20
5.4.3 Unit testing objectives 20
5.4.4 Unit testing activity tasks 21
5.4.5 Tools used for unit testing activity 21
5.4.6 Unit testing activity transition criteria 21
5.4.7 Unit testing activity closing action 21
5.4.8 Unit testing activity verification 21
5.5 Integration testing activity 22
5.5.1 Integration testing activity parameters 22
5.5.2 Integration testing objectives 22
5.5.3 Integration testing activity tasks 22
5.5.4 Tools used for integration activity 22
5.5.5 Integration testing activity transition criteria 22
5.5.6 Integration testing activity closing action 23
5.5.7 Integration testing activity verification 23
5.6 Traceability 23
5.6.1 Traceability Objectives 23
5.6.2 Traceability activity tasks 24
5.6.3 Tools used for traceability activity 24
6 Software Development Environment 25
6.1 Software development environments 25
6.2 Continuous integration environment 25
6.3 Reporting environment 25

6.3.1 Meetings minutes....25
6.3.2 Verification and validation reports....25
6.3.3 Sprint reports....25

INDEX OF FIGURES

Figure 1. Relationship with other plans....6
Figure 2. Top level project organization....9
Figure 3. MARTe development activities flow diagram....13
Figure 4. Required traceability information....24

INDEX OF TABLES

Table 1 Applicable documents....7
Table 2 Reference documents....7
Table 3 Definitions, acronyms and abbreviations....7
Table 4 MARTe deliverables for each release....10
Table 5 Requirements' properties....15
Table 6 Values for the format of a requirement identifier....15

0 INTRODUCTION

0.1 Purpose and scope of the document

Under the Specific Contract no. F4E-OFC-361-06, with subject “Fast plant controller prototype”, this document is deliverable D4 MARTe Project Management Plan (PMP).

The scope of the current document is focused in the detailed description of the software development process for MARTe framework.

This document is intended for use by anyone interested in developing or expanding MARTe’s functionalities while harmonizing their contributions.

0.2 Relationship to other plans

Next figure shows the relationship between this plan and the other plans referenced in this document.

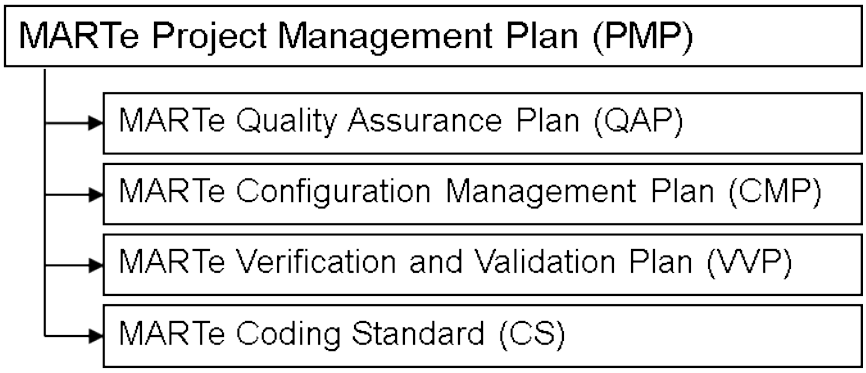

Figure 1. Relationship with other plans

Quality assurance, configuration management and verification/validation processes are known as integral processes. That is, they are present during the entire software lifecycle of MARTe.

0.3 Intellectual Property

The license to apply on source code will be EUPL v1.1 [http://ec.europa.eu/idabc/eupl.html].

0.4 Applicable and reference documents

Ref.	Code	IDM code	Description
AD1	MARTe-P-PMP-01	F4E_D_2RBL9F	(this) MARTe Project Management Plan
AD2	MARTe-P-ADD-01	F4E_D_2EXNPJ	MARTe Architecture Design Document
AD3	MARTe-P-CS-01	F4E_D_32SNQE	MARTe Coding Standard
AD4	MARTe-P-QAP-01	F4E_D_2F888J	MARTe Quality Assurance Plan
AD5	MARTe-P-CMP-01	F4E_D_2R5FL3	MARTe Configuration Management Plan
AD6	MARTe-P-VVP-01	F4E_D_3S94CT	MARTe Verification and Validation Management Plan
AD7	MARTe-P-TUT-01		MARTe Tutorials

Ref.	Code	IDM code	Description
AD8	MARTe-P-DEN-01	F4E_D_2SB7R8	MARTe Development Environment
AD9	MARTe-P-CIE-01	F4E_D_2JEP9G	MARTe Continuous Integration Environment
AD10	MARTe-P-POR-01	F4E_D_2PTQNM	MARTe Portability
AD11	MARTe-P-SRS-01		MARTe Software Requirements Specification

Table 1 Applicable documents

Ref.	Code	Date	Ver.	Description
RD1	---	2009		"EFDA–JET–CP(09)01/02 MARTe: a Multi-Platform Real-Time Framework". A. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, A. Barbalace, H. Fernandes, D.F. Valcárcel, A.J.N. Batista and JET-EFDA Contributors.

Table 2 Reference documents

0.5 Definitions, acronyms and abbreviations

Term	Definition
AD	Applicable Document
API	Application Program Interface
Backlog review	In this project, it refers to a periodical meeting, used to update the requirements or functionalities to be implemented (equivalent in this case to User Stories); to prioritize them and to assign them a comparative degree of complexity (points of user story).
ITER	International Thermonuclear Experimental Reactor
F4E	Fusion for Energy
IDM	ITER Document Management system
MARTe	Multithreaded Application Real-Time executor
RD	Reference Document
UML	Unified Modelling Language
SCC	Source Code Control
SysML	Systems Modeling Language
Sprint planning	The Sprint Planning Meeting is where the Scrum Team and Product owner determine which features and tasks will be attempted in the upcoming sprint.
Sprint review	Each sprint concludes with a Sprint Review where the team demonstrates a potentially shippable product increment.
Technical debt	In this project, it will be understood as all the minor activities which, eventually, may remain pending after the closing of a sprint.

Table 3 Definitions, acronyms and abbreviations

1 PROJECT OBJECTIVES AND CONSTRAINTS

1.1 MARTe project objective

MARTe is a multi-platform C++ real-time control middleware, with a simulink-like way of describing the problem. It allows for a modular development and execution environment to control systems, which ensures and monitors real-time, and facilitates test and commissioning.

The project objective is to develop a MARTe2 version, which will be the result of a reduction exercise of the core framework based on the lessons learned from MARTe. This version will incorporate and implement an integral quality assurance process for the development of the framework (e.g. unit tests, coding standard …).

For further information, refer to [RD1].

1.2 MARTe project scope

The MARTe project scope is limited to all activities of the MARTe team that allows the correct development of documentation and source code of MARTe modules.

1.3 Technical description

This document describes the project organization, with special emphasis in the development activities that the MARTe project needs to establish for the success of project. In this way, some chapter describes how the documentation must be established and some chapters describe all the activities that the team members must perform in the development of the project.

2 PROJECT ORGANIZATION

2.1 Organization

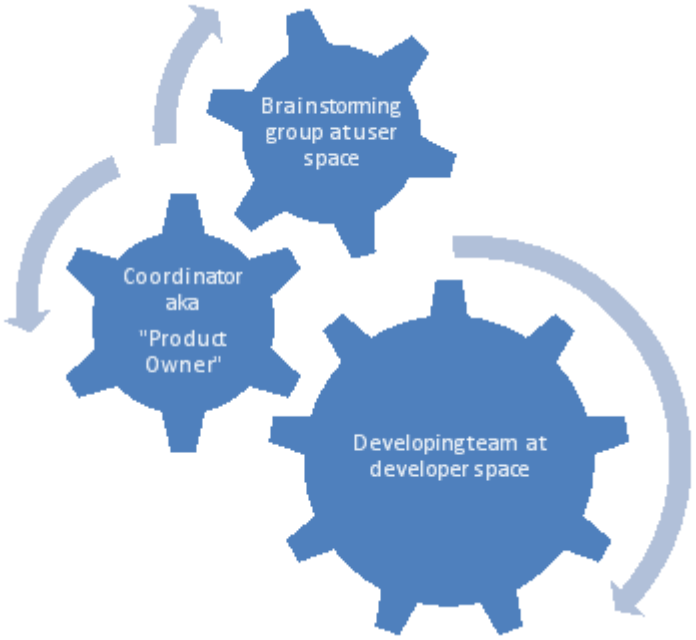

Figure 2. Top level project organization

Responsibilities:

- The brainstorming group will discuss and propose requirements for MARTe.
- The coordinator will decide which requirements are feasible for MARTe and will integrate them into the developing documentation.
- The developing team will design and implement the software accordingly to the requirements of MARTe.

2.2 Communication & Contact

The interfacing person for communication and coordination is André Neto from Fusion For Energy. His contact details are as follows:

André Cabrita Neto

Andre.Neto@f4e.europa.eu

+34 93 320 1222

3 CONFIGURATION AND DOCUMENTATION

3.1 Configuration Management

The configuration management for MARTe project is described in the following document: "*01.ES-F4E-OFC-361-06 D4 MARTe QA30 Configuration Management Plan.docx*".

3.2 Documentation Management

3.2.1 Deliverables

The next table describes all deliverables that the MARTe project will supply for each release.

Deliverable	Deliverable name scheme	Type
Requirements	MARTe requirements (vX.Y)	Report
Architecture & design	MARTe architecture & design (vX.Y)	Report
Software		
Source code	MARTe source code (vX.Y)	Source
API documentation	MARTe API documentation (vX.Y)	Report
Unit tests	MARTe unit tests (vX.Y)	Source
Integration tests	MARTe integration tests (vX.Y)	Source
QA audit	MARTe QA-audit (vX.Y)	Report
Traceability matrix	MARTe traceability matrix (vX.Y)	Report

Table 4 MARTe deliverables for each release
where X denotes a major version and Y a minor version.

The requirements deliverable describes every feature that the framework must meet and its constraints. The language used in the requirements must be clear, unambiguous, and easy to check. It is advisable to structure requirements by fields of interest, so they will be easier to understand and maintain. This deliverable is also commonly known as "Software Requirements Specification".

The architecture & design deliverable describes the system architecture and its design. It will contain a model of the main blocks of the framework as well the expected usual interactions; patterns used, and so on. The requirements must be traced backwards from the architecture and design artefacts, in order to assure that all the requirements are taken into account. If the classes modelled in the design do not have a 1 to 1 mapping with the source code classes, then it will be also necessary to trace design classes towards its implementation classes.

The software deliverable will comprise the source code and API documentation, as well the unit tests and integration tests made for the source code.

The QA audit deliverable contains all the quality reviews made for the other deliverables, as described into the AD6document. These reviews report the results and the non-conformities detected.

The traceability matrix will show the traceability between requirements and design classes. If classes modelled in the design do not have a 1 to 1 mapping with the implementation classes, and traces between design classes and implementation classes have been added, then a second matrix should be needed, but in this project is not expected to.

Versioning of deliverables: The suffix "(vX.Y)" in the name of each deliverable indicates the version of the release to which the deliverable refers, being "X" the major version and "Y" the minor version. Special attention must be paid to the fact that the deliverables versions must not evolve independently between them, instead they refer to the applicable release version of the whole project.

Types of deliverables: There will be two kinds of deliverables: reports and source. Reports will be documents created from UML models or from QA tools, while source means source code. The MARTe concept of report corresponds to F4E's IDM concept of "project file specification".

4 STANDARDS

The purpose of the standards is to define the rules and constrains to be followed during each phase of the software development process. The software verification process uses these standards as a basis of checklists for evaluating the compliance of actual outputs of a process with intended inputs.

The following sections specify the applicable standards in the software development activities of MARTe framework.

4.1 Software Coding Standard

The software coding standard applicable in the development of MARTe framework is the document referenced as follows: AD3.

The main objective of the software coding standard is to improve coding quality by applying simple practices, defining the programming languages, methods, rules and tools to be used to code the software.

This document shall enable software components of MARTe framework or related set of products to be uniformly coded. Its purpose is to define the rules, constraints and strategy for the software manual coding and integration processes. The final goal of this process is to produce well-coded source files.

All the code created during this project shall be compliant with the guidelines and rules described in the above mentioned document.

This document will serve as well as a basis for the definition of checklist that shall be used for code process verification purposes. The complete set of checklists to be used for code process verification purposes is referenced in AD6.

5 SOFTWARE ACTIVITIES

The following figure depicts a detailed flow diagram applicable to the software development process:

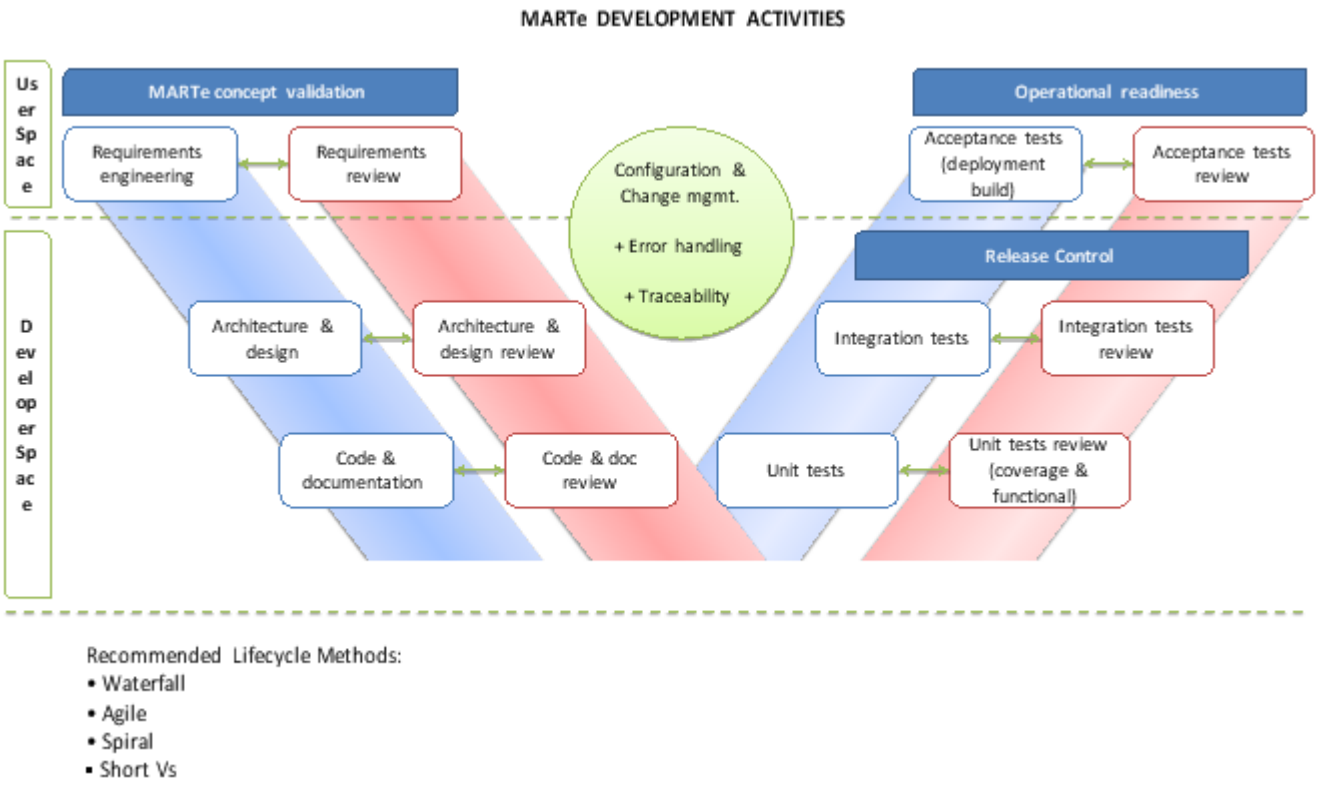

Figure 3. MARTe development activities flow diagram

As observed in the figure above, the software activities are structured following a double-V-model approach where the first V (blue) would be in charge of building MARTe framework while the second (red) would be in the MARTe quality assurance. The double-V emphasises that each of the activities have its traceable mirrored one.

The following sections of this chapter are intended to provide further details about the software development activities involved in the left side of the V shape (in blue), while the verification and validation activities from the right side (in red) will be deeply detailed in AD6.

These activities shall be respected no matter the methodology applied by the contributing organization or development group. This means that all the activities must always exist and be traced.

5.1 Requirements activity

The requirements activity translates the needs and objectives for MARTe, as defined by the stakeholders, into a proper set of requirements. These requirements will define mainly what the framework should do and which constraints it should observe.

5.1.1 Requirements activity parameters

Inputs:

The inputs for this activity are the stakeholders' needs. These needs can come from different sources with different formats, but they are the only considered input.

Outputs:

The output is a model of requirements expressed in UML (*), which can be grouped into fields of interest using UML packages. Although the output product of this activity is a model, the requirements deliverable will be a report created by exporting all the requirements and putting them into the placeholders of a report template.

(*) The requirements model is actually part of SysML metamodel, but the standard and common tools allow mixing it smoothly with UML.

5.1.2 Requirements quality attributes

The wording of the requirements shall be a complete sentence, with a subject and a predicate (usually a verb). The subject is an actor, a stakeholder, the system under development, or a design entity that is related to the requirement. The predicate specifies a condition, action, or intended result that is done for, by, with, or to the subject.

Remark: The requirements should not dictate design solutions, but sometimes some constraints are actually design solutions, so in these cases a design requirement will arise but it shall be marked as "design".

5.1.3 Requirements objectives

The final goal of the requirements activity is to develop of a set of requirements that fulfils the needs of the stakeholders and obey the expected quality attributes.

If the requirements activity is integrated into an incremental development lifecycle, then each execution of the activity will add new requirements or refine existing ones from a selected subset of stakeholders' needs.

5.1.4 Requirements activity tasks

The main tasks to execute into the requirements activity are summarized in the following list:

- Pass-through of simple needs as requirements, if they are sufficiently atomic. This means to create the requirements directly from the needs, somehow in raw mode.
- Breakdown of complex stakeholders' needs into simpler ones and passing them through as requirements, if they are atomic enough.
- Refactoring of equivalent needs as common ones and passing them through as requirements, if they are sufficiently atomic.

Other tasks to carry on into the requirements activity are listed next:

- Rewriting of requirements to assure that they fulfil the expected quality attributes.
- Quantification of requirements, if possible, and definition of tolerances where applicable.
- Correction/refactoring of duplications and inconsistencies.

5.1.5 Tools used for requirements activity

Enterprise Architect (by Sparx Systems) will be used as the tool for creating the requirements model itself. The requirements deliverable will also be created with Enterprise Architect using its built-in capabilities for generating reports from UML elements.

5.1.6 Requirements activity transition criteria

The following sections describe the transition criteria to be considered for entry/exit in the requirements activity.

5.1.6.1 Transition criteria for entry into requirements activity

The requirements activity shall begin once there are stable inputs to process. The conditions that can trigger it are:

- A new set of stakeholders' needs are ready to be transformed into requirements.
- Changes on stakeholders' needs that must be translated into existing or new requirements.
- Quality assurance has detected problems on existing requirements which have to be corrected.
- Design and implementation issues need modifications to be made on requirements.

5.1.6.2 Transition criteria for exit from requirements activity

Requirements activity will be accomplished when the selected set of stakeholders' needs have been expressed as requirements and these requirements satisfy the expected quality attributes.

5.1.7 Requirements' properties

Each requirement will have a common set of properties, some built-in and others as tag values.

Property	Type of property	Use of the property	Values of the property
Name	Built-in property	The requirement identifier	MARTe-XX-Y-Z.Z.Z(2)
Alias	Built-in property	The requirement title	Shall be one sentence
Status	Built-in property	The status of the requirement	{Proposed; Approved (1); Cancelled}
Notes	Built-in property	The full description of the requirement	Shall be a paragraph
Validation	Tagged value	The kind of validation applicable	{Test; Inspection}

Table 5 Requirements' properties

(1) Approved in this context refers to technical approval by the Coordinator.

(2) The format of a requirement identifier is "MARTe-XX-Y-Z.Z.Z", where:

Item	Values
XX	{EX: Existent; NE: New; PN: Partially new}
Y	{G: General; F: Functional; P: Performance; T: Test; D: Design}
Z.Z.Z	Hierarchical code of the requirement up to three levels (requirements with only one or two levels do not need to add trailing zeroes).

Table 6 Values for the format of a requirement identifier

5.1.8 Requirements activity closing action

At the end of the requirements activity, two tasks must be done:

- Create a baseline for the UML requirements model on Enterprise Architect.
- Export the UML requirements model on Enterprise Architect as a report.

5.1.9 Requirements activity verification

The requirements activity will have a mirror activity which will be in charge of verifying that all requirements have been correctly written and defined. This activity is described into the document “MARTe Verification and Validation Plan” (AD6).

5.2 Architecture & design activity

The architecture & design activity uses the requirements defined into the requirements activity, in order to design a framework compliant with the needs expressed in the requirements. The design will include a subset of classes considered the architectural foundation of the framework (identified as the architecture) and the rest of the classes needed for the framework (identified as the design).

5.2.1 Architecture & design activity parameters

Inputs:

The inputs for this activity are the requirements of the requirements model. These requirements can come in a structured hierarchy and are the only considered input.

Outputs:

The output is a model of classes expressed in UML, which can be grouped into namespaces using UML packages. Although the output product of this activity is a model, the architecture & design deliverable will be a report, created by exporting all the classes and putting them into the placeholders of a report template. This deliverable will clearly differentiate the architecture from the rest of the design classes.

The classes of the model shall also contain traces to requirements, which will allow knowing which classes realize each requirement. All the requirements considered in the input shall have at least one class who realize it.

5.2.2 Architecture & design quality attributes

The architecture & design will take into account the best practices and well established patterns, such as high cohesion, low coupling, and so on, to design the classes and their interactions.

5.2.3 Architecture & design objectives

The final goal of the architecture & design activity is to design a set of classes that fulfils the needs established by the requirements. This set of classes will be the design of the framework and will clearly identify a subset of them as the architecture and the others as the design.

If the architecture & design activity is integrated into an incremental development lifecycle, then each execution of the activity will add new classes or refine existing ones from a selected subset of requirements.

5.2.4 Architecture & design activity tasks

The main tasks to execute into the architecture & design activity are summarized in the following list:

- Consider the functional requirements as responsibilities and assign them as methods to the right classes, creating new classes whenever is needed.
- Consider the non-functional requirements as conditions to be met by the classes' methods and document them with interaction diagrams, state diagrams, etc.
- Add dependency traces from classes to requirements in order to trace which class realise each requirement.

Other tasks to carry on into the requirements activity are listed next:

- Redesign/refactor classes and methods to assure that they fulfil the expected quality attributes.
- Correct/refactor duplications and inconsistencies.

Optional tasks that can be postponed to the coding & documentation activity and put directly into the source code:

- Document each class with a description and an invariant.
- Document each class method with a description, a precondition/postcondition and a description for each parameter.

5.2.5 Tools used for architecture & design activity

Enterprise Architect (by Sparx Systems) will be used as the tool for creating the architecture & design model itself. The architecture & design deliverable will be also created with Enterprise Architect using its built-in capabilities for generating reports from UML elements.

5.2.6 Architecture & design activity transition criteria

The following sections describe the transition criteria to be considered for entry/exit in the architecture & design activity.

5.2.6.1 Transition criteria for entry into architecture & design activity

The architecture & design activity shall begin once there are stable inputs to process. The conditions that can trigger it are:

- A new set of requirements are ready to be realized with design classes.
- Changes on requirements that must be translated into existing or new design classes.
- Quality assurance has detected problems on existing design classes which have to be corrected.
- Implementation issues need modifications to be made on design classes.

5.2.6.2 Transition criteria for exit from architecture & design activity

Architecture & design activity will be accomplished when the selected set of requirements have been realized with design classes, these classes satisfy the expected quality attributes and they contain the traces to the requirements that they realize.

5.2.7 Architecture & design activity closing action

At the end of the architecture & design activity, two tasks must be done:

- Create a baseline for the UML architecture & design model on Enterprise Architect.
- Export the UML architecture & design model on Enterprise Architect as a report.

5.2.8 Architecture & design activity verification

The architecture & design activity will have a mirror activity which will be in charge of verifying that all classes have been correctly written and defined. This activity is described into the document “MARTe Verification and Validation Plan” (AD6).

5.3 Coding & documentation activity

The coding & documentation activity uses the design classes defined into the architecture & design activity, in order to implement them with a programming language into a set of a files, known as the source code files. These classes will also contain structured comments that, in conjunction with the classes themselves, will constitute the documentation of the framework.

5.3.1 Coding & documentation activity parameters

Inputs:

The inputs for this activity are the design classes of the architecture & design model. These design classes are the only considered input.

Outputs:

The output is a cohesive set of source code files which implement the design classes of the input in the target programming language. These files can be grouped using file system folders, following the UML packages hierarchy, while for its contained classes it is mandatory to replicate the same hierarchy of namespaces, if any. Moreover, these source files will be accompanied with make files, which will be used to generate the binary files from the source files.

In addition of the source code files which will be itself the code deliverable, the documentation deliverable will be a report created by exporting all the structured comments of the source file classes and putting them into the placeholders of a report template.

These implementation classes will implicitly trace the design classes of the design model, because it is assumed a 1 to 1 mapping between them.

5.3.2 Coding & documentation quality attributes

The codification & documentation will take into account the rules dictated by the MARTe Coding Standard [AD3], which will gather conventions and idioms to apply to source code.

5.3.3 Coding & documentation objectives

The final goal of the coding & documentation activity is to implement all the classes of the design model into a programming language embedding all the needed structured comments to document them.

If the coding & documentation activity is integrated into an incremental development lifecycle, then each execution of the activity will add new implementation classes or refine existing ones from a selected subset of the design classes.

5.3.4 Coding & documentation activity tasks

The main tasks to execute into the coding & documentation activity are summarized in the following list:

- Implement each design class and all its methods as an implementation class with the target programming language.
- Pass-through the documentation of each design class to its correspondent implementation class. If this information is not available at the design class, then create it now.
- Pass-through the documentation each design class method to its correspondent implementation class method. If this information is not available at the design class method, then create it now.

Other tasks to carry on into the requirements activity are listed next:

- Redesign/refactor classes and methods to assure that they fulfil the expected quality attributes.
- Correct/refactor duplications and inconsistencies.

5.3.5 Tools used for coding & documentation activity

Eclipse IDE for C&C++ Developers will be used as the tool for editing the source code files (both coding & documentation), while compiler toolchains will be used to build binaries from them. These toolchains can be executed from inside the Eclipse IDE or standalone from the command line.

To generate the API documentation, Doxygen will be used for extracting the classes and methods alongside its tagged structured comments.

Further details about versions of the above mentioned tool are reflected in the MARTe QA11 Development Environment [AD8].

5.3.6 Coding & documentation activity transition criteria

The following sections describe the transition criteria to be considered for entry/exit in the coding & documentation activity.

5.3.6.1 Transition criteria for entry into coding & documentation activity

The coding & documentation activity shall begin once there are stable inputs to process. The conditions that can trigger it are:

- A new set of design classes are ready to be implemented as implementation classes in the target programming language.
- Changes on design classes that must be translated into existing or new implementation classes.

- Quality assurance has detected problems on existing implementation classes which have to be corrected.
- Testing issues need modifications to be made on implementation classes.

5.3.6.2 Transition criteria to exit from coding & documentation activity

Coding & documentation activity will be accomplished when the selected set of design classes have been implemented with a programming language and these implementation classes satisfy the expected quality attributes.

5.3.7 Coding & documentation activity closing action

At the end of the coding & documentation activity, two tasks must be done:

- Create a tag for the source code files on the SCC.
- Export the API documentation from source code using Doxygen.

5.3.8 Coding & documentation activity verification

The coding & documentation activity will have a mirror activity which will be in charge of verifying that all classes have been correctly written and defined. This activity is described into the document “MARTe Verification and Validation Plan” (AD6).

5.4 Unit testing activity

The unit testing activity consists of unit tests for MARTe classes.

5.4.1 Unit testing activity parameters

Inputs:

The inputs for this activity are the implementation classes of the source code. These classes are the only considered input.

Outputs:

The output is a set of classes which implement each one a unit test for a target implementation class.

These unit test classes will implicitly trace the implementation classes of the source code, because it is assumed that each unit test class tests only one implementation class.

5.4.2 Unit testing quality attributes

All implementation classes must have a unit test which will realise at least one procedure for each class method.

The procedures of the unit tests shall test all the representative combinations of methods’ calls, which will be determined by the types and number of arguments.

5.4.3 Unit testing objectives

The final goal of the unit testing activity is to implement all the unit tests for the implementation classes of the source code.

If the unit testing activity is integrated into an incremental development lifecycle, then each execution of the activity will add new unit tests or refine existing ones for a selected subset of the implementation classes.

5.4.4 Unit testing activity tasks

The main tasks to execute into the unit testing activity are summarized in the following list:

- Implement a unit test for each implementation class of the source code.
- Implement at least one unit test procedure for each method of the implementation class (*).

(*) The only exception to this rule applies to the destructor, because it makes the tested object to be not valid and thus not queryable. Nevertheless, if the object allocates/releases external resources, then a test procedure could be considered for checking the deallocation of resources.

Other tasks to carry on into the requirements activity are listed next:

- Redesign/refactor unit tests to assure that they fulfil the expected quality attributes.
- Correct/refactor duplications and inconsistencies.

5.4.5 Tools used for unit testing activity

GoogleTest will be used as the framework to implement and execute the unit tests.

5.4.6 Unit testing activity transition criteria

The following sections describe the transition criteria to be considered to enter/exit the unit testing activity.

5.4.6.1 Transition criteria to enter into unit testing activity

The unit testing activity shall begin once there are stable inputs to process. The conditions that can trigger it are:

- A new set of implementation classes are ready to be tested.
- Changes on implementation classes that must be translated into existing or new unit tests.
- Quality assurance has detected problems on existing unit tests which have to be corrected.

5.4.6.2 Transition criteria to exit from unit testing activity

Unit testing activity will be accomplished when the selected set of implementation classes have their corresponding unit tests implemented and these unit tests satisfy the expected quality attributes.

5.4.7 Unit testing activity closing action

At the end of the unit testing activity, create a tag for the unit test source files on the SCC.

5.4.8 Unit testing activity verification

The unit testing activity will have a mirror activity which will be in charge of verifying that all unit tests have been correctly written and defined. This activity is described into the document “MARTe Verification and Validation Plan” [AD6].

5.5 Integration testing activity

The integration testing activity consists of integration tests for the architectural classes.

5.5.1 Integration testing activity parameters

Inputs:

The inputs for this activity are the implementation classes of the source code, although diagrams from the class model can help to understand their interactions. These classes are the only considered input.

Outputs:

The output is a set of classes which implement each one an integration test for a subset of implementation classes.

5.5.2 Integration testing objectives

The final goal of the integration testing activity is to implement the integration tests for the architectural classes of the source code.

If the integration testing activity is integrated into an incremental development lifecycle, then each execution of the activity will add new integration tests or refine existing ones for a selected subset of the implementation classes.

5.5.3 Integration testing activity tasks

The main tasks to execute into the integration testing activity are summarized in the following list:

- Implement an integration test for each significant architectural set of implementation classes of the source code.

Other tasks to carry on into the requirements activity are listed next:

- Redesign/refactor integration tests to assure that they fulfil the expected quality attributes.
- Correct/refactor duplications and inconsistencies.

5.5.4 Tools used for integration activity

At the time of writing, GoogleTest is thought to be used as the framework to implement the integration tests. Some of the tests for the more advanced components might use MARTe itself, as for example a GAM testing infrastructure to do some of the more complex tests to validate other GAMs.

5.5.5 Integration testing activity transition criteria

The following sections describe the transition criteria to be considered to enter/exit the integration testing activity.

5.5.5.1 Transition criteria to enter into integration testing activity

The integrations testing activity shall begin once there are stable inputs to process. The conditions that can trigger it are:

- A new set of architectural significant implementation classes are ready to be tested.
- Changes on architectural significant implementation classes that must be translated into existing or new integration tests.
- Quality assurance has detected problems on existing integration tests which have to be corrected.

Remark: Although the purpose of this activity is to test the integration between architectural significant classes, unit tests on these classes should be done before, in order to prevent that errors on implementation could affect expected integration results.

5.5.5.2 Transition criteria to exit from integration testing activity

Integration testing activity will be accomplished when the selected set of architectural significant implementation classes have their corresponding integration tests implemented and these integration tests satisfy the expected quality attributes.

5.5.6 Integration testing activity closing action

At the end of the integration testing activity, create a tag for the integration test source files on the SCC.

5.5.7 Integration testing activity verification

The integration testing activity will have a mirror activity which will be in charge of verifying that all integration tests have been correctly written and defined. This activity is described into the document “MARTe Verification and Validation Plan” [AD6].

5.6 Traceability

5.6.1 Traceability Objectives

The primary objective of traceability is to ensure that all the requirements for MARTe framework are covered by the final development.

The following figure summarizes the traceability information between deliverables that shall be progressively added, according to the progress of the different software development activities.

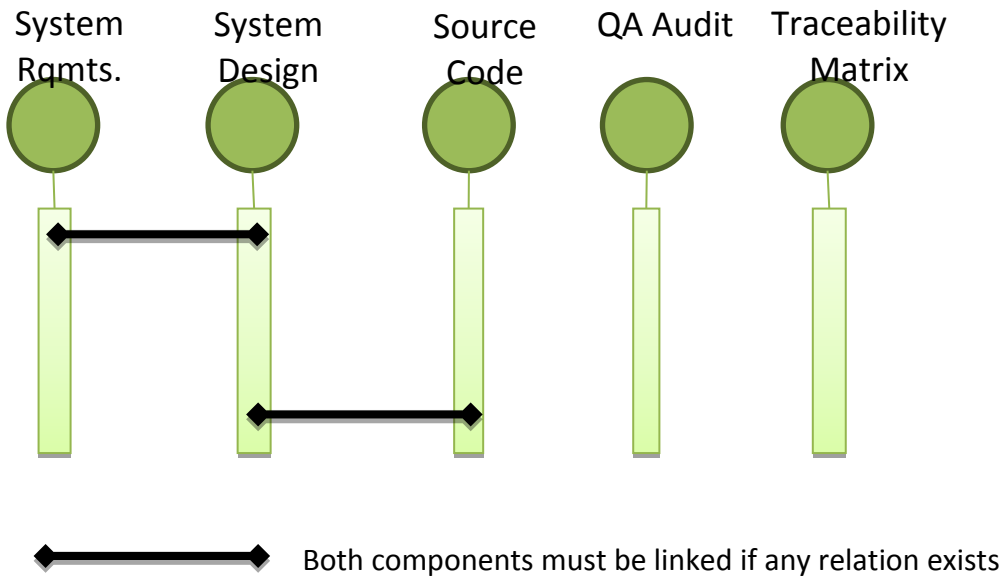

Figure 4. Required traceability information

5.6.2 Traceability activity tasks

Traceability guidance includes:

- Traceability between requirements and architecture/design. This is mandatory.
- Traceability between architecture/design and source code. This is optional if design classes have a 1 to 1 mapping to source code classes.

The main activity related with traceability will be the creation and maintenance of the trace dependencies between artefacts. This dependencies shall be easy to export as a matrix, which is the format expected to put into the traceability deliverable.

5.6.3 Tools used for traceability activity

Enterprise Architect will be used as the tool to create the trace dependencies between artefacts. This can be created adding dependencies between artefacts into a diagram or by means of the traceability matrix option of Enterprise Architect

The traceability matrix deliverable will be also created with Enterprise Architect using its built-in capabilities for generating reports from UML elements.

For large amounts of dependencies it can be difficult to maintain all of them. For that reason, GTD will use a plugin for Enterprise Architect which simplifies the creation and maintenance of theses traces. This plugin has been implemented in house by GTD in the context of the Task Order 04, but the intellectual property belongs to F4E.

6 SOFTWARE DEVELOPMENT ENVIRONMENT

6.1 Software development environments

Refer to [AD8] *document for the software development environment.*

6.2 Continuous integration environment

Refer to [AD9] *document for the continuous integration environment.*

6.3 Reporting environment

All the management and QA reports will be stored as Redmine issues as follows in next subchapters.

6.3.1 Meetings minutes

A meeting issue will be created for each meeting's minutes, where a subtask will be created for each action of the meeting.

6.3.2 Verification and validation reports

A meeting issue will be created for each verification and validation report (refer to [AD6]).

6.3.3 Sprint reports

A meeting issue will be created for each sprint report, where the following information will be put into the description field of the issue:

- Sprint planning
- Sprint review
- Technical debt
- Backlog review (restimation, reprioritization, weighting)