Programme: **E-ELT**

Project: **ELT MCAO Construction – MAORY**

MAORY Instrumentation Software Quality Assurance Plan

Document Number: E-MAO-PS0-INA-PLA-002

Document Version: 1.2

Document Type: PLA

Released On: 2022-11-28

Owner:	Andrea Balestra		28/11/2022
Approved by PI:	Paolo Ciliegi		28/11/2022
Released by PM:	Ugo Di Giammatteo		28/11/2022
	Name	Signature	Date

Authors

Name	Affiliation
Andrea Balestra	INAF - OAPd
Rosanna Sordo	INAF - OAPd

Change Record from previous version

Date	Affected Section(s)	Changes / Reason / Remarks
2018-05-25	All	First draft issue for internal review
2020-09-15	All	Adapted to new MAORY document template
	3.6	Critical item control and dependability added
2019-10-31	Table 1	Updated Instrument Software QA Role names
		Improved critical item paragraph
2020-04-30	All	New headers and minor fixes in the text.
2020-09-09	Cover and header	IS0 -> PS0
	2.2 5.2	Added applicable document ESO-288431
	All	Jira -> JIRA
2021-01-28	Cover	Added Galway logo
		Implemented PSI comments:
	3.2	Reporting will be done at progress meetings
	5.5.1	Quality target reports to be presented at progress meetings
2021-01-29		First Release for PDR
2021-11-30		PDR AI 963: • Added AD E-ELT Linux Installation Guide. • Added AD GitLab Usage Guidelines • Added paragraph (3.7.1.1) on DevEnv usage • Added code coverage metric (5.5.2.11)

		• Added possibility for subcontracors to waive DevEnv usage. • Added reference to ESO guidelines for Git workflow. Consortium workflow TBD for FDR.

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to describe the Quality Assurance Plan for the MAORY ICS Software. This document has been firstly presented at PDR and further versions shall be presented for approval at following milestones/reviews.

1.2 Scope

This document defines the Quality Assurance Plan for the MAORY ICS Software only. It covers only the Control Software part of the Instrument Project. The overall Project Quality Assurance Plan is described in [RD1].

1.3 Definitions, Acronyms and Abbreviations

AI	Action Item
ATP	Acceptance Test Plan
CI	Continuous Integration
CM	Configuration Management
CP	Common Path
CPU	Central Processing Unit
DCS	Detector Control Software
DET	Detector
DPM	Data Product Manager
E-ELT	ESO Extremely Large Telescope
FB	Function Block
FCS	Function Control Software
FDR	Final Design Review
FCS	Function Control Software
GUI	Graphical User Interface
ICS	Instrument Control System
IEEE	Institute of Electrical and Electronics Engineers
INS	ICS Software
ISDD	ICS Software Design Description
ISFS	ICS Software Functional Specification
ISMP	ICS Software Management Plan
ISUMM	ICS Software User and Maintenance Manual
ISURS	ICS Software User Requirements

LAN	Local Area Network
MAIT	Manufacturing, Assembly, Integration and Test
MAORY	Multi conjugate Adaptive Optics RelaY
MS	Maintenance Software
OCM	Observation Control Manager
OCS	Observation Control Software
PAC	Provisional Acceptance Chile
PAE	Preliminary Acceptance Europe
PDR	Preliminary Design Review
PI	Principal Investigator
PLC	Programmable Logic Controller
SDLC	Software Development Life Cycle
SPR	Software Problem Report
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
QA	Quality Assurance
TBC	To Be Confirmed
TBD	To Be Defined
TDCS	Technical Detector Control Software
WBS	Work Breakdown Structure
WP	Work Package
WS	Workstation

2. Related Documents

2.1 Applicable Documents

The following applicable documents form a part of the present document to the extent specified herein. In the event of conflict between applicable documents and the content of the present document, the content of the present document shall be taken as superseding.

AD1 MAORY Instrumentation Software Management Plan; E-MAO-IS0-INA-PLA-001 Version 3

AD2 Software Assurance Requirements for E-ELT Contracts; ESO-224035 Version 1

AD3 Control System Development Standards; ESO-193358 Version 6

AD4 ELT Instrument Control System Common Requirements; ESO-264642 Version 2.19

AD5 Guide to Developing Software for the EELT; ESO-288431 Version 2

AD6 E-ELT Linux Installation Guide; ESO-287339 Version 3

AD7 ELT ICS - GitLab Usage Guidelines; ESO-380356 Version 1.2

2.2 Reference Documents

The following documents, of the exact version shown herein, are listed as background references only. They are not to be construed as a binding complement to the present document.

RD1 MAORY Product Assurance Plan; E-MAO-000-INA-PLA-003 Version 1

RD2 Space product assurance – Software product assurance; ECSS-Q-ST-80C Rev.1

RD3 Space product assurance – Critical-item control; ECSS-Q-ST-10-04C 31 July 2008

RD4 Space product assurance – Software dependability and safety; ECSS-Q-HB-80-03A Rev.1

RD5 Space product assurance - Software metrication programme definition and implementation; ECSS-Q-HB-80-04A 30 March 2011

RD6 Space product assurance – Dependability; ECSS-Q-ST-30C Rev.1 15 February 2017

3. Software Product Assurance Programme Implementation and Management

3.1 Organization and responsibility

Software Quality Assurance (SQA) is a formal process for evaluating and documenting the quality of the work products produced during each stage of the Software Development Lifecycle (SDLC). The primary objective of the SQA process is to ensure the delivery of work products according to stated requirements and established standards. The present document is mainly based on [AD2] but it is also inspired on the practices suggested in [RD2]. Product metrics are largely inspired by practices suggested in [RD5].

The SQA responsible is in charge to define the appropriate product assurance plan and must ensure that the SW team is aware of the plan.

The software product assurance activities include:

- to participate in the definition and evaluation of the software development standards, methodologies and procedures, which shall be applied to the project;
- to check that the software and its related products conform to the adopted standards and regulations;
- to control the consistency, completeness, correctness, safety and reliability of the software;
- to control that all processes used to develop and maintain the software are appropriate, sufficient, planned, reviewed, and implemented according to the product assurance plan;
- to ensure that the configuration management related activities are correctly conducted throughout the whole SLDC;
- to ensure traceability throughout all phases of the SLDC;
- to establish and maintain a system for collecting and using the software metrics.

MAORY ICS SDLC is described in 4.1. Quality assurance will be an integral part of life cycle running in parallel with development activities. The SQA responsible shall be present in all parts of the development and will oversee all verification and validation activities that will ensure the quality of the final product.

For what concerns the tools and the framework to use, we shall use as far as possible the infrastructure that ESO will make available.

The following table defines the SQA roles and responsibilities of the members of the project team and their function at project checkpoints/milestones.

Role	Name	Responsibility	Milestone Function
SQA Manager	A. Balestra	Manages the Quality Assurance process.	Approve
System Engineer	M. Riva	Helps define product quality expectations at system level.	Approve
Project QE	E. Giro	Helps define product quality expectations at system level.	Approve
SW Project Manager	B. Salasnich	Ensures implementation of quality activities. Coordinates resolution of issues. Provides regular and timely communications.	Conduct

Table 1 - Instrument Software QA Roles and Responsibilities

3.2 Progress and Problems reporting

For general reporting, see [RD1]. For what concerns specifically SQA, the status of software product assurance program implementation will be reported at project progress meetings.

For each review and delivery milestone the assessment of the current quality of the product and processes, based on measured properties, with reference to the adopted metrics, will be reported.

3.3 Procurement, sub-contractors and supplier control

Procurement and all activities related handling of contractors is defined at project level and described in [RD1].

3.4 Risk Management

Risk management is described in [RD1], software risks will be treated in the global scope of the project, i.e. Software risk management will not be described here in isolation but will be treated as described in the MAORY Risk Management Plan [RD1]. Software risks will be listed with all other project risks in the MAORY Risk Register.

3.5 Quality Model

Software quality models will be used to specify the software product quality requirements and to monitor the software process quality. The product quality model is derived from the model specified in [RD5]. Characteristics and sub-characteristics of the quality model are listed hereafter.

3.5.1 Functionality

The capability of the software product to provide functions which meets stated and implied needs when the software is used under specified conditions.

3.5.1.1 Completeness

The capability of the software to provide full implementation of the functions required.

3.5.1.2 Correctness

The degree to which a system or component is free of faults in its specification, design and implementation.

3.5.2 Reliability

The capability of the software product to maintain a specified level of performance when used under specified conditions.

3.5.2.1 Reliability Evidence

The capability to show that software reliability analysis and assessment have been performed during the software development process.

3.5.3 Maintainability

Ability of an item under given conditions of use, to be retained in, or restored to, a state in which it can perform a required function, when maintenance is performed under given conditions and using stated procedures and resources.

3.5.3.1 Modularity

The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

3.5.3.2 Testability

Extent to which an objective and feasible test can be designed to determine whether a requirement is met.

3.5.3.3 Complexity

the degree to which a system’s design or code is difficult to understand because of numerous components or relationships among components.

3.5.3.4 User Documentation Quality

Those attributes of the software that determine the adequacy of the documentation related to software development, maintenance and operation.

3.6 Critical Item Control and Software Dependability and Safety

To perform a criticality analysis, the following check-list taken from RD3, clause C.4, has been used:

- Software items whose performances could be difficult to obtain.
- Software items not observable after integration in equipment.
- Software items not modifiable in the operational environment.
- Software items with strong intrinsic complexity.
- Software development tools with limited maintenance with respect to mission lifetime.

Based on this check-list, no critical software item has been identified. Moreover, the ICS software would be classified as Criticality category D because:

- In accordance to Table 5-1 in [RD6] ICS belongs to Severity category 3
- In accordance to Clause 5.4 in [RD6] its critical level is III
- According to [R-ICS-355] in [AD4] "The ICS high-level software shall not be involved in safety critical systems except to display the status of alarm signals." Therefore, no safety hazard can be caused by software.

Even if there is no critical software, to improve the robustness of the ICS the practices suggested in [RD4] concerning "Engineering methods and techniques supporting software dependability and safety" (Clause 6.5), "Software availability and maintainability techniques " (Clause 6.6), "Software failure propagation prevention" (Clause 6.7) will be used as guidelines. Finally, a "Defensive programming" style as defined in [RD4] Clause 6.8 shall be used as far as possible.

3.7 Tools and Supporting Environment

3.7.1 Methods and Tools

It is foreseen to adopt the tooling set as described by ESO. Therefore [AD3] is applicable, in particular the requirements described in chapter 3 (Introduction), 4 (Infrastructure Standards), 5 (Interface Standards), 6 (Notational Standards), 7 (Implementation) and 8 (Tools).

In addition to the tools described there, it is planned to use Cameo as modelling tool. Also, an internal tool (e.g. Redmine) may be used for internal configuration control of code not yet released for which the ESO standard tool (JIRA) would be used.

3.7.1.1 Development Environment

In the framework of software development for the E-ELT, ESO enforces the use of a development environment (DevEnv, see [AD6]) providing all tools necessary to the development of the code to be used for the E-ELT. The SQA responsible will verify the correct adoption of it. Subcontractors are also bound to the use of this environment. However, if the subcontractor demonstrates that the use of the DevEnv is not possible (e.g. because it violates internal development rules of the company) the SQA responsible will then only require compliance to the quality targets defined in this plan.

3.8 Assessment and Improvement Process

The SQA responsible will monitor the effectiveness and efficiency of the processes used during the development of the software according to the process metrics. However, no formal assessment and improvement process nor internal audit are foreseen. In any case, ESO shall have the right to perform its own audits in agreement with the project.

4. Software Engineering Process

4.1 Software Development Life-cycle

The ESO typical development process for instruments foresees a waterfall-like approach in the initial stage where a design phase is confined between two major milestones: the PDR and the FDR. For what concerns software, no actual development (apart from prototyping) is foreseen between these two milestones. The consequence of such procedure is that after FDR a well understood set of requirements and a sound design are in the hands of the software developers. This model, from FDR on, is therefore well suited for an incremental development process.

Incremental development is a strategy in which only a set of functionalities are implemented for each release, adding more functionalities at each delivery. Each release of the software must be usable and testable. A critical part of this strategy is clearly the division of requirements for each build to have consistent (and testable) releases and must be carefully dealt with by developers in cooperation with the SQA responsible.

Initially, it is foreseen to have a first period of “pure development” in which the base SW functionalities (e.g. functions control) will be implemented. Then, when subsystems will be integrated, it is foreseen to deliver “incremental” SW releases which will contain the functionalities needed for sub-system integration and testing. Development at the level of coordination software is expected to continue during this period. The first “feature complete” SW release will be produced for system AIT and then tested and refined during the PAE process.

4.2 Software Configuration Management

Software will be put under configuration control since the very start of the development. ESO guidelines for Git use shall be used (see [AD7]), including naming conventions. The implementation in the MAORY Consortium context of the Git workflow is still TBD and shall be detailed and presented at FDR.

The SQA responsible shall analyse the impact of any change requested to the code. Any change request to released code shall be made using the standard configuration control tool (i.e. JIRA).

Major releases shall be issued for each major milestone. Minor releases may be delivered when a coherent set of functionalities is implemented and tested. In any case, it is required that a release is issued at least every 3 months (see [AD2] R-IDP-22). The SQA responsible will always oversee releases.

4.3 Documentation and Related Processes

MAORY INS documentation will be produced, following the plan outlined in [AD1].

5. Software Product Quality Assurance

5.1 Quality Requirements

All requirements are listed in the ISURS document; therefore, no additional quality requirement is to be found here. However, the SQA responsible will make sure that all requirements related to performances and to quality in general are grouped together in the relevant document(s) (e.g. the compliance matrix, see [AD1]) and appropriate tests are foreseen.

5.2 Verification and Validation of Requirements

Verifying user requirements at the end of the PDR stage will establish the proper basis for initiating the Software Design stage activities. The User Requirements Document must contain, at a minimum, documentation on the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces.

5.2.1 Verification

Verification activities (i.e. controlling that everything is being done in the right way) shall be implemented by interaction of the SQA responsible with the developers, formalized in meetings to be held regularly. A set of automated tools will be run during CI runs to assess that metrics are adequately covered and verified.

In addition, the following activities will be performed by the SQA responsible as part of requirements verification:

- Produce a traceability matrix tracing all user requirements back to system objectives and forward to Software Design elements.
- Evaluate user requirements and relationships for correctness, consistency, completeness, accuracy, readability and testability.
- Assess how well the requirements satisfy the system objectives.
- Assess the criticality of requirements to identify key performance or critical areas of software.
- Review internally documents produced.
- Verify execution of tests, check and keep under configuration control test reports.

5.2.2 Validation

Validation (i.e. controlling that the right thing is being done) shall be carried out by the SQA responsible mainly by means of functional tests. Where possible, tests against a hardware model shall be performed. In any case the SQA responsible will ensure that the software product implements the agreed design and satisfies the given requirements. To help in this tracing activity, the usage of Cameo and its requirement management plugin is foreseen.

In addition, the following activities will be performed as part of requirements validation:

- Plan acceptance testing, including criteria for:
    - compliance with all requirements
    - adequacy of user documentation
    - performance validation
- Plan documentation of test tasks and results.
- Execute the Acceptance Test Plan.
- Document acceptance test results in the Acceptance Test Report.

5.3 Code control and test

In general, MAORY ICS SW will be tested following the practices defined by ESO (see [AD5]).

The plan foresees to monitor the code quality using ESO infrastructure. It is expected to use the continuous integration system provided by ESO (i.e. Jenkins and integrated tools) to build and test the code, to calculate code metrics and to check coding standards. This process should be run checking on a regular basis for changes in the repository and running the tests. Output of the runs shall be delivered to the developers involved and to the SQA responsible who will archive them for further analysis.

During sub-system and system AIT, testing of software on the real hardware will be performed following dedicated test plans, usually limited to a sub-set of the software.

Dynamic tests (e.g. unit tests) shall be performed regularly following ESO guidelines.

Monitoring of the tests will be performed through test reports.

5.4 Problem reporting and corrective action

During development, integration and testing, problems will be reported using an internal issue tracking system (see 3.7.1). Open issues will be checked periodically (e.g. on a bi-weekly basis) to ensure they're followed up. After PAE or PAC, ESO's JIRA system will be used. Issues will be closed only after verification by the originator or the SW PM or SQA.

5.5 Metrication Programme

A metrication programme is described hereafter based on input from [AD2] and [RD5].

5.5.1 Process Metrics

The SQA responsible will regularly assess that the life cycle phases duration is in accordance with the project schedule.

The SQA responsible will keep track of all problems detected during verification and validation activities analysing trends that may negatively impact the project.

The SQA responsible will periodically inspect and analyse the output of the continuous integration tests (i.e. compilation, static analysis, unit tests etc.) to assess how well the development process is doing. The same will be done with the test reports. Any suspicious trend or result will be reported to the PM. Reports will be presented at project progress meetings.

All output of tests, both in the form of logs of CI system and test reports will be archived and kept under configuration control.

5.5.2 Product Metrics

The following list of metrics has been produced based on chapter A.3.3 of [RD5] and according to the format defined there adapted to the MAORY project.

5.5.2.1 Requirement allocation

Main Characteristic	Functionality
Sub Characteristic	Completeness
Metric name	Requirement allocation
Goal	This metric identifies the relationship among: - Higher level requirements and software level requirements; - SW requirements and SW design.
Owner / Producer	Owner: SW Project Manager Producer: Development Team
Target audience	SW Project Manager, SQA responsible
Evaluation method	Traceability matrices.
Formula	X= A/B, where: A = number of system level requirements for software that have one or more trace to SW requirements or SW design components; B = number of system level requirements for software
Interpretation of measured value	0 <= X <= 1, the closer to 1 the better; any number < 1 should be justified.
Life cycle phase	Collected during software related system engineering, SW requirements &

	architecture engineering, SW design & implementation engineering processes. Provided at PDR, and updated as required.

5.5.2.2 Requirement implementation coverage

Main Characteristic	Functionality
Sub Characteristic	Completeness
Metric name	Requirement implementation coverage
Goal	This metric provides the percentage of requirements that are implemented and properly verified in the product.
Owner / Producer	Owner: SQA Responsible Producer: Development Team
Target audience	SQA Responsible, SW Project Manager
Evaluation method	Analysis of traceability matrices (i.e. requirements versus verification method), eventually with the aid of some automatic tools (e.g. MagicDraw).
Formula	X= A/B, where: A = number of correctly implemented requirements confirmed by verification (including test, inspection, review, analysis, validation); B = number of requirements
Interpretation of measured value	0 <= X <= 1, the closer to 1 the better Those requirements that are not implemented or verified should be documented.
Life cycle phase	Collected during SW validation and verification processes. Provided at PDR, and updated afterwards as required.

5.5.2.3 Requirement completeness

Main Characteristic	Functionality
Sub Characteristic	Completeness
Metric name	Requirement completeness
Goal	This metric provides the number of remaining open points in the requirements.
Owner / Producer	Owner: SQA Responsible Producer: Development Team
Target audience	SQA Responsible, SW Project Manager
Evaluation method	Analysis of requirements, possibly with the aid of automatic tools (e.g. macros).
Formula	X= A/B, where: A= number of requirements containing TBCs/TBDs; B= total number of requirements;
Interpretation of measured value	0 <= X <= 1, the closer to 0 the better; any number > 0 should be justified.
Life cycle phase	Collected during SW requirements & architecture engineering. Provided at PDR, and updated afterwards as required.

5.5.2.4 SPR trend analysis

Main Characteristic	Functionality
Sub Characteristic	Correctness
Metric name	SPR trend analysis
Goal	This metric provides a graphical representation of the evolution of SPR correction over time, classified by SPR severity levels.
Owner / Producer	Owner: SQA Responsible Producer: CM responsible
Target audience	SQA Responsible, SW Project Manager, , CM Responsible
Evaluation method	Analysis of SPR database.

Formula	There is no formula associated to this metric.
Interpretation of measured value	- The graphic exhibit convergence between number of raised and fixed problems. The lower the gap between these numbers the better. - At a certain point during testing, the number of new discovered problems should remain stable. This can be used as test completion criteria.
Life cycle phase	Collected during SW verification and validation, SW delivery and acceptance, SW operation and SW maintenance processes. Provided at FDR and updated afterwards.

5.5.2.5 Adherence to coding standards

Main Characteristic	Functionality
Sub Characteristic	Correctness
Metric name	Adherence to coding standards
Goal	This metric provides a subjective assessment of the adherence to the applicable coding standards.
Owner / Producer	Owner: SQA Responsible Producer: Development Team
Target audience	SQA Responsible, SW Project Manager
Evaluation method	Coding standards checklist, to be filled in with the help of static analysis tools. - Rules covered by automatic tools are expected to be verified for 100% of the code. - Sample code size will be specified for those rules to be verified manually (at least 5% of the code should be inspected).
Formula	X = number of violations

Interpretation of measured value	The closer to 0 the better; violations should be documented in the code.
Life cycle phase	Collected during SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.5.2.6 Cyclomatic complexity (VG)

Main Characteristic	Maintainability
Sub Characteristic	Complexity
Metric name	Cyclomatic complexity
Goal	This metric provides an indication of the code complexity, based on the number of linearly independent test paths for each subroutine.
Owner / Producer	Owner: SW Project Manager Producer: Development Team
Target audience	SW PA manager, SW Project Manager, V&V leader
Evaluation method	Static code analysis with the support of automatic tools.
Formula	The cyclomatic complexity of a single routine (function or procedure) is defined as: VG = (number of edges) - (number of nodes) + 2 Then, the cyclomatic complexity of a module is defined as: X = average VG for all routines in the module
Interpretation of measured value	In general, the lower the cyclomatic complexity the simpler and more testable a software product is. However, low complexity at routine level can increase granularity at design level. Therefore, a good compromise is to be reached for both properties. VG target value: 20

	There may be exceptions to this threshold (such as multiple-choice statements or error-handling code).
Life cycle phase	Collected during SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.5.2.7 Nesting level

Main Characteristic	Maintainability
Sub Characteristic	Complexity
Metric name	Nesting level
Goal	This metric provides an indication of the code complexity, based on the depth of imbrications of the code.
Owner / Producer	Owner: SW Project Manager Producer: Development Team
Target audience	SQA Responsible, SW Project Manager
Evaluation method	Static code analysis with the support of automatic tools.
Formula	The nesting level of a single routine (function or procedure) is defined as: NL = maximum number of nested statements (simple or multiple-choice decisions, loops) in the routine Then, the nesting level of a module is defined as: X = maximum NL for all routines in the module
Interpretation of measured value	In general, the lower the nesting level the simpler and more testable a software product is. However, too plain code might imply too complex design. Therefore, a good compromise should also be reached in this case.

	NL target value: 7 Exceptions are possible (e.g. error-handling code) but should be documented.
Life cycle phase	Collected during SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.5.2.8 Lines of code (LOC)

Main Characteristic	Maintainability
Sub Characteristic	Complexity
Metric name	Lines of code
Goal	This metric provides an indication of the code complexity, based on the number of executable lines per routine.
Owner / Producer	Owner: SW Project Manager Producer: Development Team
Target audience	SW Project Manager, SQA Responsible,
Evaluation method	Static code analysis with the support of automatic tools.
Formula	LOC = (total number of lines of code) – (comment and blank lines)
Interpretation of measured value	This metric should be computed at routine/class level. LOC target value: 100
Life cycle phase	Collected during SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.5.2.9 Comments frequency

Main Characteristic	Maintainability
Sub Characteristic	Complexity

Metric name	Comment frequency
Goal	This metric provides an indication of the legibility of the code in terms of percentage of comment lines.
Owner / Producer	Owner: SW PA manager Producer: Development Team
Target audience	SW PA manager, SW Project Manager
Evaluation method	Static code analysis with the support of automatic tools.
Formula	X= A/B, where: A = number of comment lines (excluding headers); B = LOC + (number of lines of comments) = total number of lines excluding blank lines
Interpretation of measured value	0 <= X <= 0.3 Target value: 0.2
Life cycle phase	Collected during SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.5.2.10 Requirement testability

Main Characteristic	Maintainability
Sub Characteristic	Testability
Metric name	Requirement testability
Goal	This metric provides the percentage of requirements that are verified by test.
Owner / Producer	Owner: SQA responsible Producer: SQA responsible
Target audience	SQA responsible, SW Project Manager
Evaluation method	Analysis of traceability matrices (i.e. requirements versus validation tests), eventually with the aid of some automatic tools (e.g. Cameo).

Formula	X = A/B, where: A = number of requirements verified by test; B = total number of requirements
Interpretation of measured value	0 <= X <= 1, the closer to 1 the better; any number > 0 should be justified. A value below 0.8 should be a matter of concern.
Life cycle phase	Collected during SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.5.2.11 Code coverage

Main Characteristic	Reliability
Sub Characteristic	Reliability Evidence
Metric name	Line Coverage
Goal	This metric determines how much of the code structure was executed by the requirements-based tests.
Owner / Producer	Owner: SQA responsible Producer: SQA responsible
Target audience	SQA responsible, SW Project Manager
Evaluation method	Dynamic analysis of the code with the support of automatic tools (e.g. Polyspace and/or DevEnv tools).
Formula	X = A/B, where: A = number of executed statements/decisions/conditions; B = total number of statements/decisions/conditions.
Interpretation of measured value	0 ≤ X ≤ 1 , the greatest value is better; the minimum accepted value is 50% with a goal of 100%
Life cycle phase	Collected during SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.5.2.12 SPR status

Main Characteristic	Reliability
Sub Characteristic	Reliability evidence
Metric name	SPR status
Goal	This metric provides a snapshot of the SPR status at a given point in time, classified by SPR severity levels.
Owner / Producer	Owner: SQA responsible Producer: CM responsible
Target audience	SW Project Manager, SQA Responsible.
Evaluation method	Analysis of SPR database.
Formula	There is no formula associated to this metric.
Interpretation of measured value	- No major/critical SPR should remain open at PAE.
Life cycle phase	Collected during SW verification and validation, SW delivery and acceptance, SW operation and SW maintenance processes. Provided at FDR and updated afterwards and presented on request.

5.5.2.13 User documentation completeness

Main Characteristic	Maintainability
Sub Characteristic	User documentation quality
Metric name	User documentation completeness
Goal	This metric provides the number of remaining open points in the user documentation.
Owner / Producer	Owner: SW PA manager Producer: Development Team
Target audience	SQA Responsible, SW Project Manager
Evaluation method	Analysis of user documentation, eventually with the aid of automatic tools (e.g. macros).

Formula	X= A/B, where: A= number of sections containing TBCs/TBDs; B= total number of sections;
Interpretation of measured value	0 <= X <= 1, the closer to 0 the better; any number > 0 should be justified.
Life cycle phase	Collected during SW design and implementation engineering, SW verification and validation processes. Provided at FDR, and updated afterwards as required.

5.5.2.14 Coupling between objects

Main Characteristic	Maintainability
Sub Characteristic	Modularity
Metric name	Coupling between objects (CBO)
Goal	To measure the number of other classes to which a class is coupled.
Owner / Producer	Owner: SW Project Manager Producer: Development Team
Target audience	SQA Responsible, SW Project Manager
Evaluation method	Static code analysis with the support of automatic tools.
Formula	No Formula. Just by counting the number of distinct non-inheritance related class hierarchies on which a class depends.
Interpretation of measured value	CBO<=4 The larger the number of couples, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult
Life cycle phase	Collected during design and implementation engineering, SW validation and verification processes. Provided at FDR, and updated afterwards as required.

5.6 Software quality control

No other activity of software quality control is foreseen in addition to those described so far.

5.6.1 Definition of quality targets

Quality targets are listed in 5.5.2. Software quality targets are normally monitored during verification and validation phase and report is given in the relevant milestone.

5.7 Training

Expertise of all ICS SW staff will be assessed by interview, looking at past experiences and acquired certifications (if any). Where necessary, formal training will be required (e.g. for PLC programming, etc.).

***** End of document *****