	ROC Verification and Validation Plan	Ref: **ROC-GEN-SYS-PLN-00040-LES** Issue **02** Revision **02** Date: December 20, 2019

SOLAR ORBITER RPW ROC

ROC Verification and Validation Plan

ROC-GEN-SYS-PLN-00040-LES

Iss.02, Rev.02

Prepared by	Date	Signature
Sonny Lion RPW ground segment validation engineer		
Verified by	**Date**	**Signature**
Xavier BONNIN and Stéphane Papais RPW ground segment project and software manager		Signé numériquement par : stephane.papais@nexeya.com Nom DN : CN = stephane.papais@nexeya.com Date : 2019.12.20 14:08:20 +01'00'
Approved by	**Date**	**Signature**
Xavier BONNIN RPW ground segment project manager		Signature numérique de Xavier Bonnin DN : cn=Xavier Bonnin, o=LESIA, ou, email=xavier.bonnin@obspm.fr, c=FR Date : 2019.12.20 16:07:57 +01'00'

CLASSIFICATION PUBLIC ☒ RESTRICTED ☐

l'Observatoire de Paris LESIA
Laboratoire d'Études Spatiales et d'Instrumentation en Astrophysique

CNRS-Observatoire de PARIS
Section de MEUDON-LESIA
5,Place Jules Janssen
92195 Meudon Cedex - France

Change Record

Issue	Rev.	Date	Authors	Modifications
0	0	04/02/2016	X. Bonnin	First draft
1	0	07/11/2017	S. Lion	Sphinx conversion - validation approach - risks identification and mitigation
2	0	10/12/2018	S.Lion	Major changes of the document content
2	1	10/12/2018	S.Lion	Use Gitlab instead of JIRA
2	2	20/12/2019	S.Lion and X.Bonnin	Update list of test cases - Update test case convention - Update validation procedures - Update AD/RD issue/rev.

Acronym List

Acronym	Definition
CCSDS	Consultative Committee for Space Data Systems
CDF	Common Data Format
CUC	CCSDS Unsegmented time Code
HF	High Frequency
ICD	Interface Control Document
LF	Low Frequency
LL	Low Latency
MEB	Main Electronic Box
PA	Pre-Amplifier
RLLP	RPW Low Latency Pipeline
ROC	RPW Operation Centre
ROT	RPW Operation Toolkit
RPW	Radio and Plasma Waves instrument
SCM	Search Coil Magnetometer
SGS	Science Ground Segment
SGSE	Software Ground Support Equipment
SOC	Science Operation Centre
TDS	Time Domain Sampler
THR	Thermal Noise and High Frequency Receivers
ssh	Secure Shell
SWF	Snapshot Waveform
XML	eXtended Markup Language

Contents

List of Figures

List of Tables

1 General

1.1 Scope of the document

This document presents the verification and validation plan of the RPW Operation Centre (ROC).

The ROC verification and validation plan (RVVP) provides the definition of organizational and management approach to the implementation of the verification and validation activities for the centre.

The present document does not cover the following validation activities:

- The validation tests of the data exchange interfaces between the ROC and the Solar Orbiter Mission Operations Centre (MOC) and Science Operations Centre (SOC) [AD5], which is under the responsibility of the European Space Agency (ESA).
- The validation tests performed in the framework of the System Operation Validation (SOV) and the System Validation Test (SVT) campaigns [RD5].
- The validation of the RPW Low Latency Virtual Machine (LLVM) [RD4], which is operated by the SOC.

Besides, the verification and validation plan related to the RPW data products is described in the DVVP [RD?].

1.2 Applicable Documents

Tab. 1.1: Applicable documents

Mark	Reference/Iss/Rev	Title of the document	Authors	Date
AD1	ROC-GEN-MGT-PLN-00013-LES/1/4	ROC Project Management Plan (PMP)	Yvonne de Conchy - Xavier Bonnin	20/12/2016
AD2	ROC-GEN-SYS-PLN-00040-LES/2/3	ROC Software Development Plan	Xavier Bonnin	11/2017
AD3	ROC-GEN-OTH-NTT-00036-LES/1/0	ROC Project Glossary of terms	Xavier Bonnin	13/09/2017
AD4	ROC-GEN-MGT-QAD-00033-LES/1/3	ROC Software Assurance /Product Assurance Plan (SPAP)	Stephane Papais	26/09/2019
AD5	SOL-SGS-TS-0006/1/0	Solar Orbiter Instrument Teams - SOC Test Specification	Nana Bach - Christopher J. Watson	30/08/2017
AD6	ROC-GEN-SYS-PLN-00002-LES/2/0	ROC Concept Implementation Requirements Document (CIRD)	X.Bonnin	07/05/2019

1.3 Reference Documents

This document is based on the documents listed in the following table:

Tab. 1.2: Reference documents

Mark	Reference/Iss/Rev	Title of the document	Authors	Date
RD1	ROC-GEN-SYS-SPC-00026-LES/1/2	ROC Software System Specification (RSSS)	X.Bonnin	14/10/2019
RD2	SOL-ESC-PL-00001/1/1	Solar Orbiter Mission Implementation Plan (MIP)	I.Tanco	31/12/103
RD3	ROC-GEN-SYS-SPC-00036-LES/1/0	ROC Software System Design Document (RSSDD)	X.Bonnin	11/2017
RD4	SOL-SGS-TN-0006/1/0	SOC Engineering Guidelines for External Users (SEGU)	R.Carr	04/03/2015
RD5	SOL-ESC-IF-05010/1/2	Planning Interface Control Document (PLID)	L.Michienzi	07/2015
RD6	SOL-SGS-ICD-0006/1/1	Extended Flight Events and Communications Skeleton (E-FECS) file ICD	C.Watson	24/03/2017
RD7	SOL-SGS-ICD-0003/1/1	Solar Orbiter Instrument Operation Request Interface Control Document (IOR ICD)	C.Watson	07/03/2019
RD8	SOL-SGS-ICD-0007/1/1	Solar Orbiter Telemetry Corridor ICD	C.Watson	17/04/2019
RD9	SOL-SGS-TN-0017/0/2	SOC-Provided Ancillary Data for Solar Orbiter	A.Walsh	18/09/2017
RD10	ROC-PRO-PIP-ICD-00037-LES/1/2	RPW Calibration Software Interface Control Document (RCSICD)	M.Duarte, X.Bonnin	05/06/2019
RD11	ROC-GEN-SYS-NTT-00019-LES/2/0	ROC Engineering Guidelines for External Users (REGU)	X.Bonnin	11/2017
RD12	ROC-GEN-SYS-NTT-00008-LES/1/3	ROC Engineering Guidelines (REG)	X.Bonnin	09/11/2016
RD13	ROC-TST-OTH-NTT-00073-LES/1/0	ROC Test Plan Template	X.Bonnin	06/11/2018
RD14	ROC-GEN-SYS-URD-00064-LES/1/0	ROC User Requirements Document (URD)	RPW Team	07/05/2019
RD15	ROC-GEN-SCI-PLN-00077-LES/01/00	RPW science data verification and validation plan (DVVP)	X.Bonnin and S.Lion	27/05/2019

1.4 About this document

1.4.1 Access policy

The present document is accessible without any restriction.

Any modification of this document must be approved by the RPW Ground Segment Project Manager before publication.

1.4.2 Terminology

All terms used in this document, and which are not listed in the table below must follow the definition in [AD3].

2 Validation plan

2.1 Context and philosophy

The ROC will be in charge of supervising the ground segment for the RPW instrument on-board Solar Orbiter ESA mission.

In this context, the centre has to implement a set of tools, called ROC Software System (RSS), in order to ensure the functions of the ground segment.

Concerning the validation process for the Solar Orbiter instrument ground segments, it is assumed that:

- There will be no formal review of the instrument ground segment by ESA before the launch.
- There will be no formal validation of the instrument ground segment design by ESA, outside the scope of the interface validation tests (see the ROC Project Management Plan (PMP) [AD1] for the list of tests). Especially, the payload ground segments will be considered as “ready-for-flight” from the ESA point of view, as soon as they have successfully passed these tests.

Considering the two assumptions and as requested in the ROC Software Assurance/Product Assurance Plan (SPAP) [AD4], the ROC shall plan to validate its infrastructure and ensure the compliance with the expected top-level requirements, as defined in the ROC Concept and Implementation Requirements (CIRD) [AD6].

2.2 Definitions

2.2.1 Unit tests

Unit testing is the phase in which individual units of source code are tested to determine whether they are fit for use. The unit testing strategy is described in the section *Validation strategy*.

2.2.2 Integration tests

Integration testing is the phase in which individual software modules are combined and tested as a group to expose defects in the interfaces and in the interactions between integrated components or systems.

The integration testing strategy is described in the section *Continuous integration et non-regression strategies*.

2.2.3 Validation tests

The validation testing phase ensures that the software meets the requirements defined in the CIRD [AD6], the RSSS [RD1] and the URD [RD14]. The validation testing strategy is described in the section *Validation strategy*.

2.3 Convention

2.3.1 Validation campaign naming

The RSS Validation campaigns (RSSVC) shall be uniquely identified using the following naming convention:

ROC_RSSVC<rss-version>_V<iteration>

where:

- <rss-version> is a string identifying the RSS version (3, 4 or 5);
- <iteration> is a 2-digits integer starting from 01 and incremented by 1 for each campaign rerun.

2.4 Overall approach

2.4.1 Overview

A validation campaign shall be scheduled before each major release (see section *ROC validation activity planning overview* for details). It shall consist of running series of tests, which permit to verify the compliance with expected requirements.

A validation campaign requires to define:

- List of requirements to be validated for the given release
- List of requirement-related test cases
- People involved and expected roles
- Expected inputs and outputs (data, documentation)
- Expected environment (hardware, software)
- Detailed planning

All of this information will have to be reported in a dedicated test plan (see section *Validation campaign Test plan*).

Each test case will have to be fully described in testcard file, as explained in the section *Validation campaign Test plan*, and following the unit, integration and validation test levels, as defined in the next section.

2.4.2 Perimeter

2.4.2.1 At the ESA level

The validation of the interface between ESA centres and ROC is outside of the scope of this document.

However, as a component of the Science Ground Segment (SGS) of Solar Orbiter, the ROC is involved in validation test campaigns driven by ESA. These tests mainly concern the data exchanged between the ROC and the Solar Orbiter MOC and SOC, as well as the related interfaces. Although the ROC is not in charge, it will have to support ESA in the preparation, execution and analyzing of these tests. Additionaly, the ROC will have to deliver and collaborate with the SOC team, concerning the validation tests of the deployment and execution of the RPW LLVM [RD4] at the ESAC site (Madrid, Spain). All of the documentation related to these validation tests (i.e., test plan, test report) will be issued by ESA. Nevertheless, the ROC may write for some of these tests internal documentation to fully cover the activity at RPW-level.

2.4.2.2 At the RPW level

The validation of the instrument performance and calibration is outside of the scope of this document.

Nevertheless, the ROC will have to validate the RPW data products, as explained in the DVVP.

2.4.2.3 At the ROC level

The validation of the RSS implies the validation of its software units, with their own external/internal interfaces and data products. Besides, the ROC shall plan verifications of the RPW Calibration Software (RCS), delivered by the RPW sub-systems teams, namely: THR, TDS, LFR, SCM and Bias Lead-CoI teams.

2.4.3 Implementation

2.4.3.1 Involved software

The validation campaigns shall permit to ensure the compliance of ROC software with the specification defined in the RSSS [RD1] and URD [RD14].

It concerns at least the following software units: - ROC Operations and Data Pipeline (RODP) - RPW Calibration Software (RCS) - ROC Software Ground Support Equipment (ROC-SGSE) - Monitoring and control sub-system User Interfaces (MUSIC)

2.4.3.2 Tools, Techniques and Methods

References about tools, software and packages discussed in this section are available in Appendix B.

N.B. The version of the tools used for each campaign shall be clearly indicated in the related test plan.

At the system level, the validation relies on the following tools:

- Gitlab - A web platform used for software versioning and issue tracking. Gitlab CI application shall be also used to run continuous integration
- Docker - A container platform used to deployed and run software in an isolated environment
- locust - A python utility to do distributed load testing used to perform stress tests on the MUSIC Web tools.
- Confluence - A collaborative wiki platform.

For ROC software written in Python:

- Pytest - A framework that makes it easy to write small unit tests as well as complex functional testing.
- hypothesis - A library to parametrize tests and simply generate random data matching given specifications.

For ROC software written in Javascript:

- Mocha - A flexible test runner that can be used to run JavaScript tests on the server or in the browser.
- Chai - An assertion library, similar to Node's build in assert that can be used in browser.
- Enzyme - A JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse React Components.

N.B. Gitlab is hosted in a remote server maintained the DIO

2.4.3.3 Procedures

The following steps shall always be performed prior to each campaign:

- Issuing the test plan related to the validation campaign.
- Listing the test cases to be run for the campaign and creation of the corresponding set of testcard files.

- Listing the version of the software units to validate. Especially, the version used for the validation shall be clearly tagged on the software Git repository.
- Creation of a new campaign branch and issue label using the ROC Validation group on the Gitlab server (https://gitlab.obspm.fr/ROC/Validation), keeping only the selected test cases;
- Compilation, automatic tests and deployment of the software units on the validation environment.

Once the validation platform is deployed, the campaign takes place as follow:

- execution of the identified test cases and generation of expected data products and reports. The test cards shall be filled at this stage;
- analysis of the test results;
- generation of the validation report;
- archiving of test resources;
- cleaning of the validation platform.

If anomalies are detected, a second full or partial validation test campaign can be carried out once the investigation has been completed and corrections fixed:

- Identification of a new Git revision;
- Generation and deployment of the new version on the validation platform;
- Rerun of all test cases to ensure non regression;
- Update of the validation report.
- Archiving of the new test resources;
- Cleaning of the validation platform.

2.4.4 Expected Documentation

The ROC validation campaign documentation, i.e. test and validation reports, will have to be stored in the ROC documentation management system (DMS): https://ged.obspm.fr/j_obspm/docbase/topic/browse_classic?topicID=T357&timeID=1541493255913.

No documentation is requested by ESA concerning these campaigns.

2.4.4.1 Validation campaign Test plan

A validation campaign test plan shall be written in preparation of each validation campaign.

It shall provide:

- The test design (overview, responsibilities, environment and prerequisites). Especially, the ROC software name and version involved shall be listed in this section.

- The test organization (overview, schedule, responsibility and related reviews)
- The test definition (input data, preconditions, post-conditions, success criteria and detailed test procedure). This section shall also give the list of the test cards that will be run during the campaign.

A template file [RD13] is available in the ROC DMS.

2.4.4.2 Validation campaign test card

The test card file is the baseline document used to execute a given test case.

Each test card shall contain the following items:

- test_case-id: The Test case unique identifier (see *Test case naming* for details)
- requirements: The list of CIRD requirements that have been tested
- version: Version of the test card file
- priority: the priority of the test case related to the CIRD requirements (‘low’, ‘medium’, ‘high’ or ‘critical’)
- created_by: Author(s) of the test card file
- reviewed_by: test card file reviewer(s)
- test_case_description: Short description of the test case objectives
- test_scenario: Summary of the test case procedure
- tester_s_name: Name of the tester(s)
- date_tested: Execution date of the test case
- test_case_status: Execution status of the test case (‘Not executed’, ‘Passed’, ‘Partially passed’, ‘Failed’, ‘Suspended’)
- test_readiness_status: Readiness status of the test case (‘Not ready’, ‘Ready’, ‘Partially ready’)
- comment: Any relevant information about the test_case_status and/or test_readiness_status
- known_limitations: List of known limitations
- prerequisites: List of prerequisites prior to the test case execution
- test_data: List of test data required to run the test case
- test_steps: shall contains the list of step_details, specification_id (RSSS and URD requirements ID), expected_results, actual_results and status.

The test card shall be provided as a YMAL format file with the following naming convention:

<test-case-id>_<test-card-version>.yml

Where:

- <test-case-id> is the test case identifier, as defined in the section *Test case naming*;
- <test-card-version> is the version of the test card;

All test card files shall be saved as templates into a dedicated ROC Git repository. Each time a new validation campaign is planned, new empty copies of the test card files shall be generated and provided to the testers.

In this case, the test card copies shall be renamed as:

<test-case-id>_<test-card-version>_<rssvc-name>.xml

Where:

- <rssvc-name> is the name of the RSSVC

The testers shall use these copies during the tests. Filled test card copies shall be archived in the ROC DMS at the end of the campaign.

2.4.4.3 Validation campaign test report

A validation campaign test report shall be written after each validation campaign.

It shall provide:

- The list of completed test card files and automated reports (see section *Test Reporting*) generated during the validation tests.

2.4.5 ROC validation activity planning overview

The timeline below presents the main validation activities involving the ROC. The ROC development planning is given in the ROC Software Development Plan (SDP) [AD2].

The ROC shall organize a validation campaign before each RSS main version release; starting with the RSS3 release.

Additionally, ESA runs its own tests and validation campaigns. The overall responsibities in these campaigns is specified in sections: *Responsabilities <responsabilities>* and *Validation at ESA level <validation_esa>*.

The following table summarizes the RSS software units that will be validated during each campaign. The list of expected requirements for each RSS release can be found in [RD?].

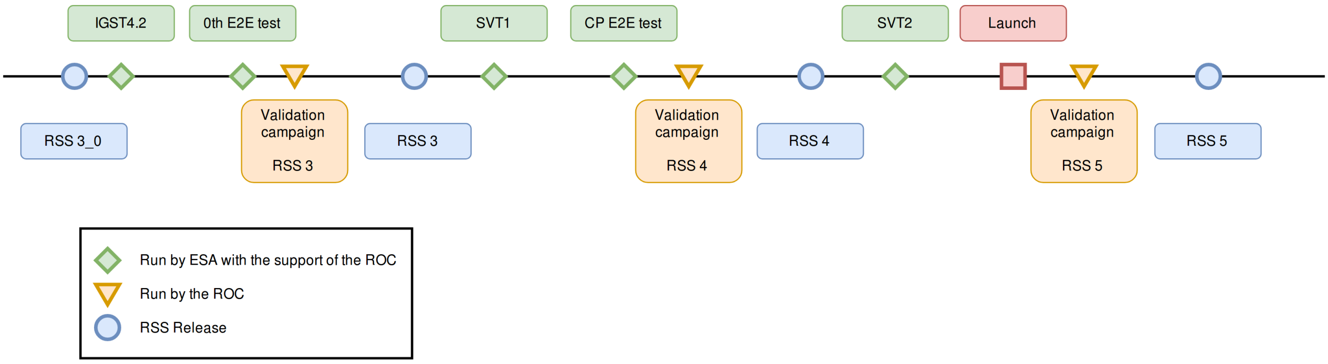

Fig. 2.1: Validation activity planning overview

Tab. 2.1: Validation campaigns planning

Validation campaign	Components to validate
RSS3 (rehearsal)	• MUSIC-FIGARO prototype version • MUSIC-FAUST prototype version • RODP preliminary version
RSS4	• MUSIC main page prototype version • MUSIC-FIGARO operational version • MUSIC-FAUST operational version • MUSIC-TV prototype version • RODP operational version at LESIA • ROC-SGSE “mission” instance at LESIA
RSS5	• MUSIC (all components) • RODP prime instance at LESIA • MUSIC-OPERA operational version (TBC) • MUSIC-SISSI prototype version (TBC) • MUSIC TV operational version • LLVM (LESIA backup instances)

2.4.5.1 Schedule for the ROC Software System validation campaigns (RSSVC)

The schedule of each RSSVC shall be detailed in the dedicated test plan.

2.4.5.2 Schedule for the ESA tests

2.4.5.2.1 Schedule related to the ROC-SOC interfaces tests

The schedule related to the SOC interfaces tests is detailed in [AD5]. A summary of main tests are provided here:

- SOC TMC and E-FECS compatibility test
- SOC IOR compatibility test
- SOC GFTS interface test

Detailed planning for each test will be provided by the SOC in the dedicated test plan.

2.4.5.2.2 Schedule related to the SOC RPW LLVM instance tests

The main milestones related to the RPW LLVM validation process with the SOC are reported in [AD1].

The ROC must thus ensure that, at each step of the SOC validation process. The LLVM testing environments at LESIA is used to verified and deliver an instance with the expected functionalities. This environment will have to be maintained during the mission, in the case where a new LLVM version needs to be delivered.

2.4.5.2.3 Schedule related to the ROC-MOC interfaces tests

The schedule related to the MOC interfaces tests is detailed in [AD?]. It concerns the following tests:

- MOC PDOR/MDOR compatibility test
- MOC DDS/GFTS interface test

Detailed planning for each test will be provided by the MOC in the dedicated test plan.

2.4.6 Resources

The validation testing environment shall rely on the development/production environments defined in the Software Development Plan [AD2]. Moreover, it shall be consistent with the technology and related design presented in the ROC Software System Design Document [RD3].

The following sections give the main hardware and software resources, which shall be used for each RSS validation campaign. Specific resources for a given validation campaign will have to be presented in the dedicated test plan.

2.4.6.1 Hardware resources

The ROC validation campaigns should be performed in an environment similar to the operational - also called "production"- environment [AD2]. Therefore, the following hardware resources shall be required:

- a Debian server identical to the production one, see [AD2] for architecture and configuration details;
- a network connexion, for software installation and database communications;
- a workstation (client computer) with the minimal configuration described in [RD1];
- a database server with PostgreSQL.

In addition, some mutualized resources are used during the validation process:

- the ROC Gitlab server or another git repository to download the software sources

2.4.6.2 Software resources

The ROC shall rely on the following main software to run the validation:

- Python (3.6)
- Node.js (8.11)
- npm (6.4)

The validation tests shall be tracked using the ROC Validation group, available in the Gitlab server managed by the Direction Informatique de l'Observatoire (DIO).

Each test campaign shall be run in a dedicated isolated workspace, using a Python virtual environment.

Input data required to perform the tests are defined in section :*2.5.3 <Datasets>*.

Part of the validation tests are to be done manually. Personnel requirements are described in section *2.2.7 <Personnel requirements>*.

Finally, the MUsIC web page requires a recent web browser, either Firefox (56.0+) or Chrome (61.0+). Both will be required for a complete test of the application performances.

2.4.7 Responsibilities

2.4.7.1 Key personnel

The key personnel of a RSSVC are:

- The RPW ground segment software validation engineer;
- The RPW ground segment project manager;
- The RPW ground segment Assurance Product Manager;

- the ROC developers (the complete list of developers is available in the ROC Project Management Plan [AD1]);
- the ROC operators.

Additional actors shall participate to the validation activities, as listed in the section *2.2.7 <Personnel requirements>*.

The detailed responsibilities shall be reported for each RSSVC in the dedicated test plan. The following sections present how the validation activities are shared between the identified actors.

2.4.7.2 Test writing

The RPW ground segment software validation engineer supervises the test implementation. In particular, he/she ensures that the tests cover all the specifications addressed in the ROC Software System Specification [RD1] and are compliant with the test guidelines.

2.4.7.2.1 Automatic tests

Tests writing is part of the development cycle [AD2]. Therefore the responsibility of a proper implementation of tests belongs to each developer of the ROC.

2.4.7.2.2 Beta testing procedures

At this stage of the project, only the MUSIC application validation requires a beta testing phase. The responsibility for developing this test procedure is shared between ROC operators and developers.

2.4.7.2.3 Verification procedures

For software tests run using the continuous integration infrastructure. The dedicated Gitlab CI interface shall be used to check the status of the tests.

Any other specific verification procedure shall be clearly defined in the test plan.

2.4.7.3 Test platform

The RPW ground segment software validation engineer is also responsible for setting up and maintaining the test platform, including the continuous integration server (see section 2.5 for details about the validation environment).

2.4.7.4 Execution and verification of tests

The RPW ground segment project manager shall ensure that the validation of the RSS meet all the specifications addressed in the CIRD [AD6].

2.4.7.5 LLVM

The ROC is in charge of:

- Validating the "backup" instance of the LLVM deployed at the ROC site;
- preparing the tests that will be run at ESA;
- ensuring the ESA LLVM instance is compliant with the expected specifications [RD? TBD]

The formal validation is the responsibility of ESA.

2.4.7.6 Instrument commanding

The ROC is in charge of:

- checking file format compliance of TC sequences;
- checking file format compliance of IOR/MDOR/PDOR;
- ensuring the TC sequences submitted to ESA are compliant with the specifications at system, command/control and interface levels [RD? TBD, DR? TBD]. A main step before delivery will be to test the TC sequences with the RPW-PI laboratory GSE.

The formal validation of TC sequences/IOR/MDOR/PDOR is the responsibility of ESA.

2.4.8 Personnel requirements

The personnel required for testing differs significantly from one RSS component to another. In particular, at this stage of the project, only MUsIC requires a beta testing phase. The table below details the needs by component/task/product:

Tab. 2.2: Personnel requirements

Component/Task/Product	Needs (see the ROC Project Management Plan [AD1] for a complete list of the developers)
Automatic tests launch and monitoring	• An engineer to run the tests • An engineer to monitor and report results
Verification and compilation of reports	Two engineers
Scientific validation of data products	TBD
TC sequences testing using the GSE	• An operator to run the GSE • An ROC engineer to analyze and report results • An GSE administrator (in backup) in case of GSE issues • An ROC administrator (in backup) in case of ROC tools (FIGARO, FAUST and ROC-SGSE) issues
MUSIC beta testing	• a member of the ROC developpement team (to follow beta-testing as a developer point of view and to report bugs) • a ROC administrator to ensure that MUSIC is up-and-running during the beta-tests • a scientist for science data visualization functional beta-tests • a ROC operator for RPW commanding functional beta-tests • a member of the RPW Operations Board (ROB) for operation planning and SBM selection functional beta-tests • a member of the RPW CNES team for RPW commissioning functional beta-tests • a member of the GIGL for security aspects

2.4.9 Risks

2.4.9.1 Risks identification

The following points of failure during validation campaigns have been identified:

Tab. 2.3: Risk matrix

Point of failure	Consequences	Impact	Probability
Network connection issues	Installation and servers communication issues	Major	Unlikely
Server down	Unable to run the RSS/Gitlab (depending on the server)	Major	Unlikely
Power failure	All the servers are off	Major	Very unlikely
Staff absence	The validation tasks will be done by another operator	Minor	Possible
Computer breakdown	The validation campaign will be done using another computer	Minor	Unlikely
Database corruption/error	Integration and acceptance tests can not be performed	Major	Very unlikely

2.4.9.2 Contingency plans

2.4.9.2.1 Network issues

If it is a short outage of the network (less than 2 days), the validation campaign will just be delayed. In case of prolonged network issues, the campaign will be postponed.

2.4.9.2.2 Power issues

If a brief power cut occurs (less than 2 days), the validation campaign will just be delayed. In case of prolonged power issues, the campaign will be postponed.

2.4.9.2.3 Server breakdown recovery

The RSS is designed to be installed in less than 30 min on any computer meeting the specification [RD1]. Moreover, the ROC has three Debian instances (`roc`, `roc-dev` and `roc-web`) that can be used temporarily as backup servers.

2.4.9.2.4 Computer breakdown recovery

The ROC has at least three computers which meets the specifications and can be used for the validation campaign.

2.4.9.2.5 Gitlab backup

2.4.9.2.5.1 Software sources and issues management

To ensure a perfect redundancy of the versioning system and an issue manager, the ROC shall:

- periodically dump the ongoing issues;
- maintain a up to date copy of all the git repository on another server.

These issues dumps and repositories shall be stored on `roc-dev`.

2.4.9.2.5.2 Unit, acceptance and regression tests

Gitlab is just automation tool. Each RSS instance comes with integrated tests which can be run independently.

2.5 Validation tasks identification

2.5.1 Test Reporting

The tests reports format depend on the test runner:

- Pytest and Mocha publish JUnit XML test reports which are collected and made available to users. These reports also include the results of coding guidelines tests as well as comment and test coverage metrics.
- Locust results are saved in a CSV file. This file includes statistics on response times and errors.
- Beta test reports shall be based on the Beta Testing Report Template (Appendix C). This document includes a series of test cases with evaluation grids and feedback sections.

Each validation campaign will be monitored using JIRA. Confluence (associated with JIRA) will be used to be compile all the results of the campaign in a single document, the ROC Software System Validation Test Report.

2.5.2 Test versioning

To keep track of the validation campaign, a dump of the pre-production database (see section 5.3), tests inputs and results shall be stored on the data storage server, `lesia11` (accessible via NFS from `roc` and `roc-dev`).

ROC databases are on a server maintained by the GIGL

2.5.3 Controls

The ROC sets up the following controls:

- Static checks to verify the compliance with applicable quality standards and allow the production of reports;
- Dynamic controls using unit tests, integration and validation test cases;
- Manual controls to validate the functionalities that can not be checked automatically, typically the HMIs.

2.5.4 Requirements monitoring

The Topcased tool (https://www.polarsys.org/topcased) shall be used to generate the ROC requirements traceability matrices between the CIRD, URD, RSSS and test case ID.

2.6 Validation strategy

2.6.1 Unit testing strategy

The strategy for unit testing is specific to each RSS component. The MUSIC [RD TBD] and Pipelines [RD TBD] test plans describe respectively the test approach for MUSIC and for the RODP/ROC-SGSE/LLVM.

2.6.2 Continuous integration et non-regression strategies

The RSS continuous integration system rely on Gitlab CI application. Fig. 2.2 represents the typical workflow of the system.

N.B. Jenkins solution tool have been replaced by Gitlab CI to ensure the ROC software automated testing.

During the development of a new feature, the developer shall send his code to the Gitlab repository. This action triggers the validation process on the Gitlab server. At first, the system runs successively unit and acceptance tests.

- If one of these steps failed (Fig. 2.2, cases 1 and 2), a negative feedback is sent to the developer.
- If the tests are completed, a positive feedback is sent and the developer shall switch to manual validation.

Once all these steps are marked as successful, the feature can be released (Fig. 2.2, case 3).

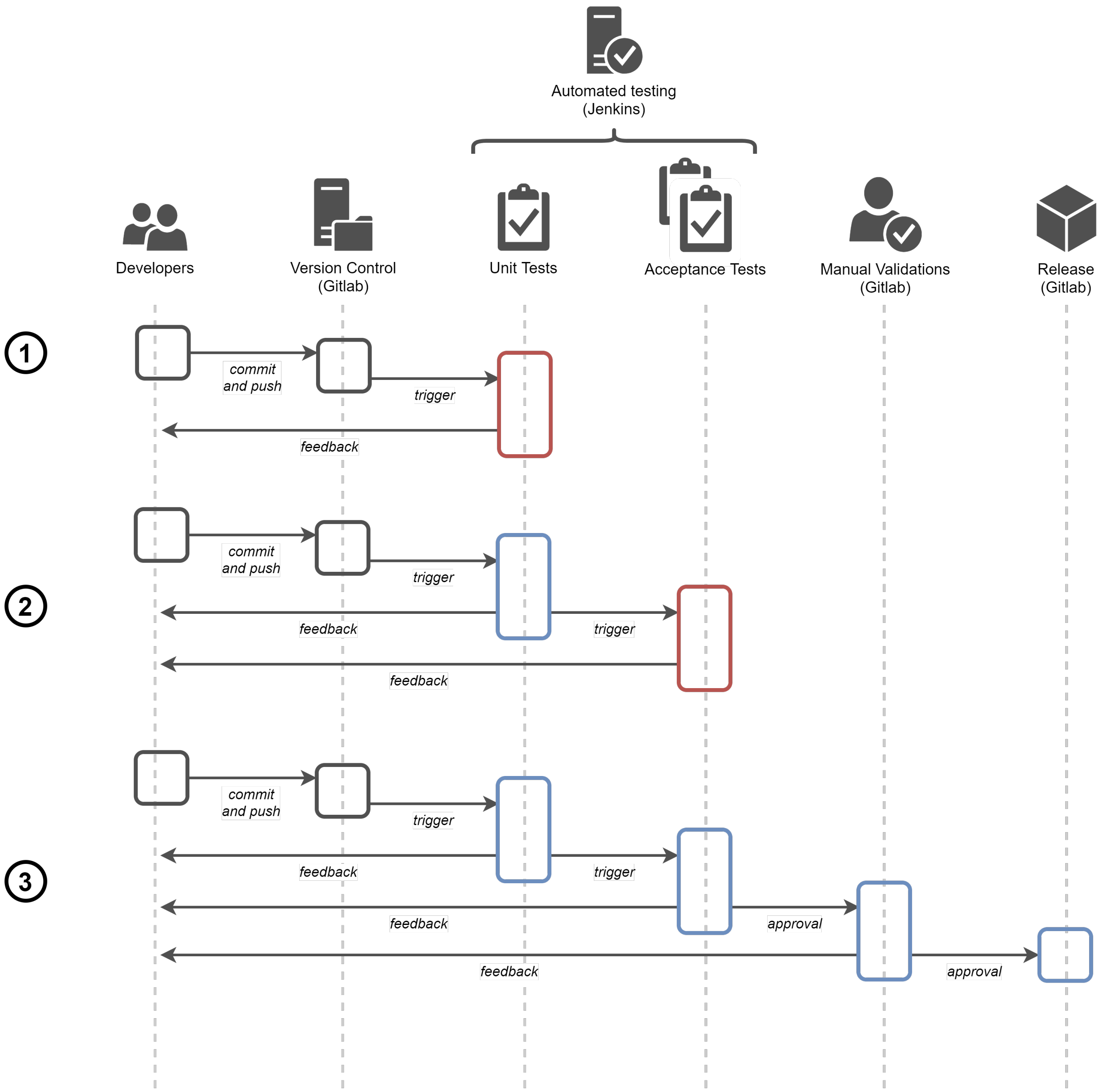

Fig. 2.2: Continuous integration flow

2.7 Validation environment

2.7.1 Continuous integration

The automated test environment is a part of the continuous integration system. Continuous integration cycles and non regression tests are described in the section 2.4.3. The Fig. 2.3 summarise the continuous integration system organisation.

N.B. Jenkins solution tool have been replaced by Gitlab CI to ensure the ROC software automated testing.

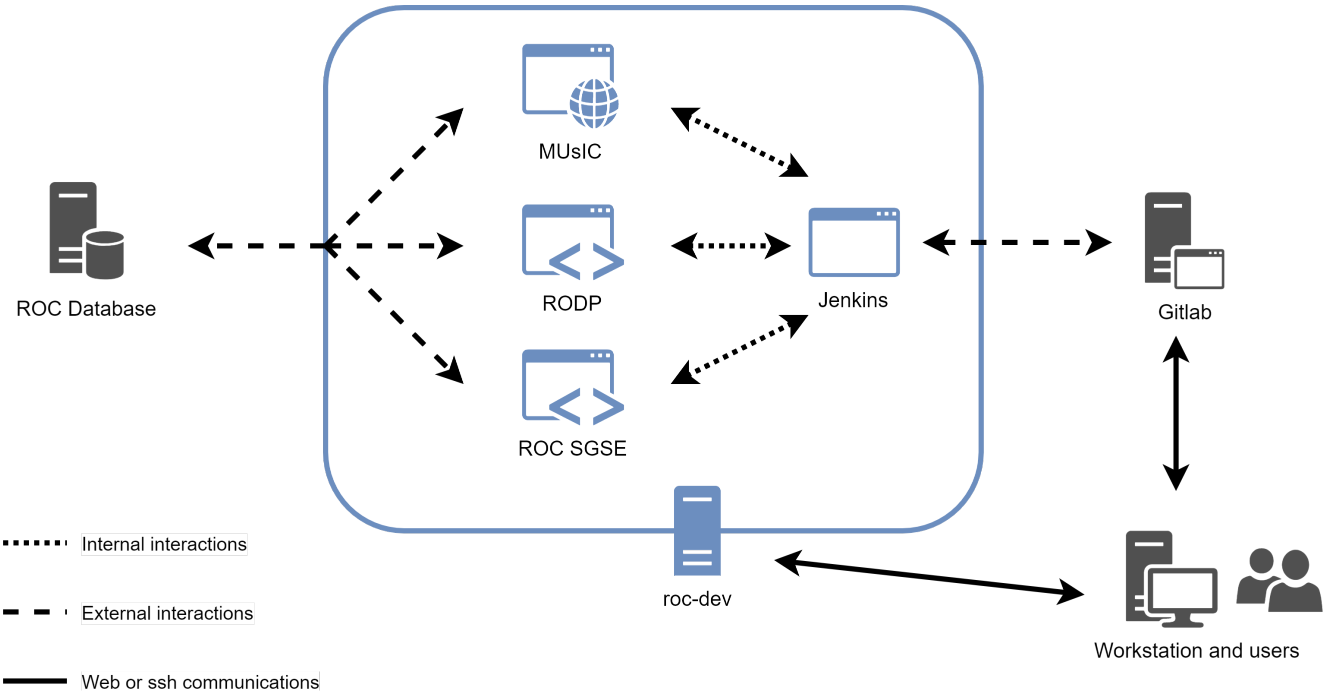

Fig. 2.3: Continuous integration environment

Gitlab runners are run on the **roc-dev** server using dedicated tokens. The testing jobs are triggered automatically by push requests but can be run manually using the Gitlab web interfaces.

Tests reports are accessible via the TBD web interface or directly by ssh. On roc-dev, the reports are stored in the TBD project workspace.

2.7.2 Validation platform

The validation is performed on the pre-production environment.

As shown in Fig. 2.4, the overall organisation of the pre-production environment relies on the continuous integration infrastructure. The only difference is that the RSS units run on their respective production servers.

Note that the test runners on the production servers use a dedicated user account to avoid interfering with the production instances.

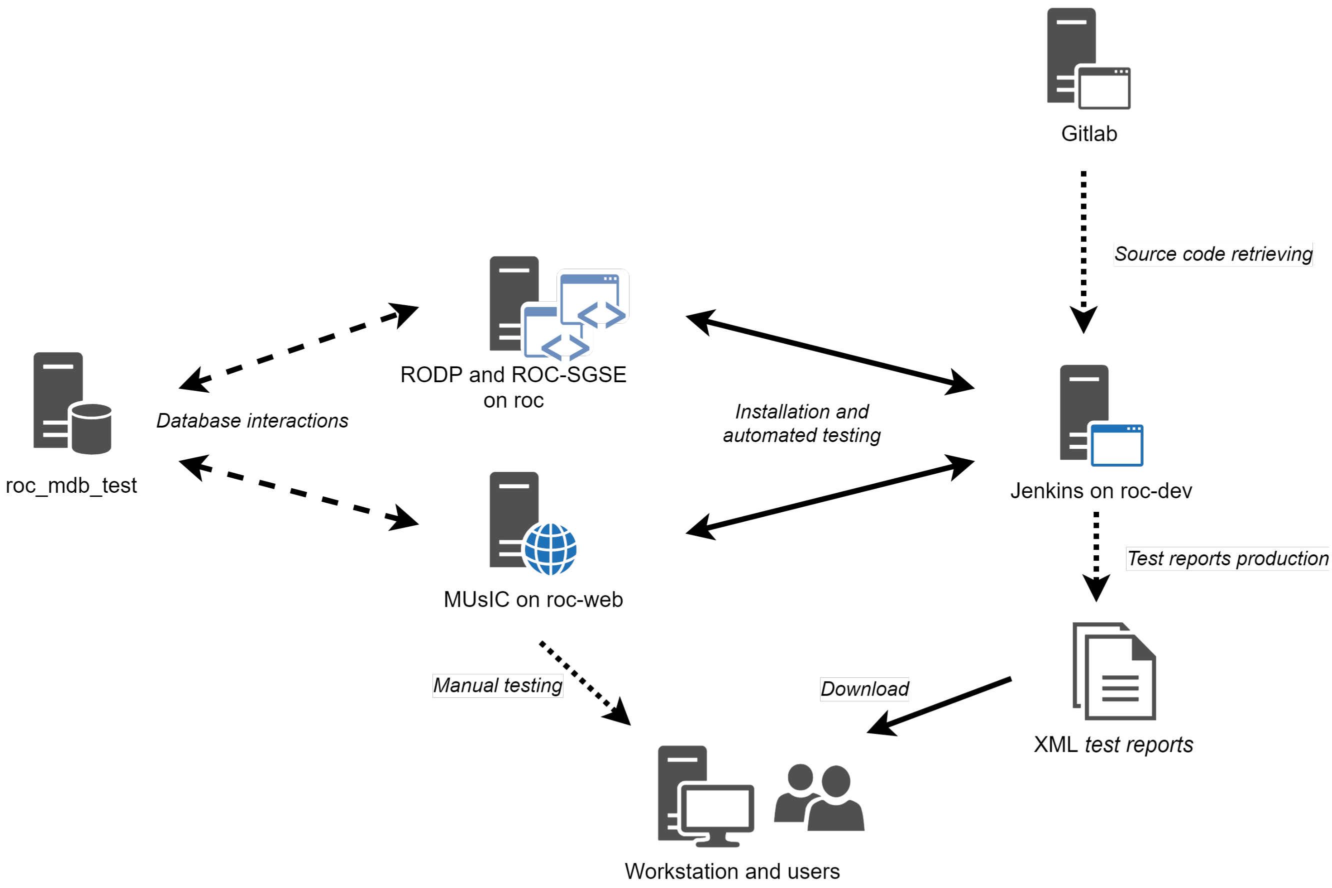

Fig. 2.4: Pre-production environment

2.7.3 Test data

The data used to perform the validation shall be listed in the test plan.

2.8 Interfaces validation

2.8.1 MOC interfaces validation

The organization related to the MOC data exchange interfaces test and validation is not fully known at this stage of the project.

Nevertheless, it is assumed that the following interfaces will be tested:

- MOC Generic File Transfer System (GFTS)

- MOC Data Dissemination System (DDS)

Especially, the DDS interface will have to be tested in conditions close to the system environment at MOC site during the commissioning.

It will also include compatibility tests of the formats of the data files sent to the MOC via the GFTS, namely: the Payload Direct Operations Request (PDOR) and the Memory Direct Operations Requests (MDOR) (see [RD5]).

The formal validation and resulting reports will be done by the MOC from the tests performed with the ROC.

2.8.2 SOC interfaces validation

The organisation and specification related to the SOC interfaces tests and validation are detailed in [RD? TBD].

The following SOC interfaces will be tested:

- SOC GFTS

It will also include compatibility tests of the formats of the data files exchanged via the GFTS between the ROC and SOC, namely: the Instrument Operation Requests (IOR) [RD6], the Telemetry Corridor (TMC) [RD8], the Extended Flight Event and Communication Skeletons (E-FECS) [RD6] and the ancillary data [RD9].

The formal validation and resulting reports will be done by the SOC from the tests performed with the ROC.

3 Verification plan

3.1 Concept and definition

The verification process can be defined as the evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition.

3.2 Control procedures

Automatic reports are exported from Gitlab. Both automated and manual test cases are then compiled in Jira and Confluence to generate the final report.

TBC (problem reporting and resolution, deviation and waiver policy, control procedures)

3.3 Identification of verification activities

The validation is based on test campaigns linked to the RSS release planning. Each validation test case follows from a specific requirement of the CIRD [AD6].

3.3.1 Test case naming

Validation test cases follow the naming convention described below:

ROC-<function>-<identifier>-<target>

Where:

- <function> is a string identifying the tested ROC function (see table below);
- <identifier> is a 3-digits integer (e.g., '010'), which shall be unique for a given <function>.
- <target> is a string given the function target

The exhaustive list of labels is given below:

Tab. 3.1: Nomenclature

Task	Label
Data retrieval	DATA_RETR
Data production	DATA_PROD
Data distribution	DATA_DIST
Data storage and archiving	DATA_ARCH
Data visualization	DATA_VISU
Instrument commanding	RPW_COM
Instrument monitoring	RPW_MONIT
Ground support	GRD_SUPPORT
ROC infrastructure monitoring	ROC_MONIT

3.3.2 Test case description

Each test case must contain the following information:

- **Test Purpose**: This paragraph describe the purpose of this test.
- **Prerequisites**: This paragraph lists all the actions that must be done before the execution of the test case.
- **Input data**: This paragraph lists all the data needed to run the test: products, auxiliary data.
- **Expected outputs**: This paragraph lists all the expected outputs of the test.
- **Reference Data**: This paragraph lists all the data useful for verifying outputs.
- **Standalone**: This paragraph indicates whether the test case is independent of other test cases.

3.3.3 Verification procedures

This section presents the list of test cases by functionality. For each test case, the list of

3.3.3.1 Data retrieval

Tab. 3.2: Data retrieval

Test case	Tested function
ROC-DATA_RETR-010-RPW	Retrieving RPW data
ROC-DATA_RETR-020-OPS	Retrieving Mission operation input data
ROC-DATA_RETR-030-ANC	Retrieving Mission ancillary data

3.3.3.2 Data production

3.3.3.2.1 Producing RPW data files

The automated verification of RPW data files is detailed in the DVVP [RD15].

The verification of the RCS is detailed in the pipelines test plan [TBD].

Tab. 3.3: RPW data files

Test case	Tested function
ROC-DATA_PROD-010-LZ	Producing RPW LZ data
ROC-DATA_PROD-020-L0	Producing RPW L0 data
ROC-DATA_PROD-030-L1	Producing RPW L1 data
ROC-DATA_PROD-040-L2	Producing RPW L2 data for Bias
ROC-DATA_PROD-041-L2	Producing RPW L2 data for LFR
ROC-DATA_PROD-042-L2	Producing RPW L2 data for SCM
ROC-DATA_PROD-043-L2	Producing RPW L2 data for TDS
ROC-DATA_PROD-044-L2	Producing RPW L2 data for THR
ROC-DATA_PROD-050-L3	Producing RPW L3 data
ROC-DATA_PROD-060-HK	Producing RPW HK “digest” data
ROC-DATA_PROD-070-QL	Producing RPW data summary plots (i.e; quick-looks)

3.3.3.2.2 Processing mission ancillary data files

As described in the CIRD [AD6], the ROC does not plan to produce any ancillary data files. Therefore no testing is planned for these files.

3.3.3.2.3 Producing RPW Low Latency data

Tab. 3.4: Low Latency data

Test case	Tested function
ROC-DATA_PROD-080-LL01	Deliverying RPW Low Latency LL01 data processing pipeline

3.3.3.2.4 Validating RPW science data

The validation of the RPW science data is outside of the scope of this document and presented in the DVVP [RD15].

3.3.3.2.5 Re-processing RPW data

Tab. 3.5: Re-processing

Test case	Tested function
ROC-DATA_PROD-090-REPROC	Re-processing RPW data

3.3.3.2.6 Converting on-board time

Tab. 3.6: On-board time

Test case	Tested function
ROC-DATA_PROD-100-OBT_UTC	Converting on-board time (OBT) to UTC time

3.3.3.3 Data dissemination

3.3.3.3.1 Distributing preliminary RPW data

Tab. 3.7: Preliminary data

Test case	Tested function
ROC-DATA_DIST-010-PRE_DATA	Distributing preliminary RPW data (part 1 - publishing data)
ROC-DATA_DIST-015-PRE_DATA	Distributing preliminary RPW data (part 2 - downloading data)

3.3.3.3.2 Distributing definitive data

As described in the CIRD [AD6], the ROC is not expected to be a public data provider for RPW. Therefore no testing is planned for the distribution of definitive data.

3.3.3.3.3 Distributing ancillary data

Tab. 3.8: Ancillary data

Test case	Tested function
ROC-DATA_DIST-020-ANC	Distributing mission ancillary data

3.3.3.4 Data storage and archiving

3.3.3.4.1 Storing data at LESIA

Tab. 3.9: Storing at LESIA

Test case	Tested function
ROC-DATA_ARCH-010-LESIA	Storing data at LESIA

3.3.3.4.2 Archiving RPW data

Tab. 3.10: Archiving RPW data

Test case	Tested function
ROC-DATA_ARCH-020-ESAC	Archiving RPW data at ESAC
ROC-DATA_ARCH-030-CDPP	Archiving RPW data at CDPP

3.3.3.5 Data visualization

Tab. 3.11: Visualizing data

Test case	Tested function
ROC-DATA_VISU-010-VISU	Visualizing data

3.3.3.6 Instrument commanding

3.3.3.6.1 Requesting Medium-Term Planning (MTP) instrument operations

Tab. 3.12: MTP instrument operations

Test case	Tested function
ROC-RPW_COM-010-MTP_PROD	Producing RPW MTP IOR
ROC-RPW_COM-020-MTP_DELIV	Submitting RPW MTP IOR to SOC
ROC-RPW_COM-030-MTP_CONST	RPW MTP IOR constraints
ROC-RPW_COM-040-RESOU	Compute RPW resources

3.3.3.6.2 Requesting Short-Term Planning (STP) instrument operations

Tab. 3.13: STP instrument operations

Test case	Tested function
ROC-RPW_COM-050-STP_PROD	Producing RPW STP IOR
ROC-RPW_COM-060-STP_DELIV	Submitting RPW STP IOR to SOC
ROC-RPW_COM-070-SBM_EVENT	Requesting SBM1/SBM2 events data

3.3.3.6.3 Requesting non-routine instrument operations

Tab. 3.14: Non-routine instrument operations

Test case	Tested function
ROC-RPW_COM-080-PDOR-PROD	Producing RPW PDOR
ROC-RPW_COM-081-PDOR-DELIV	Submitting RPW PDOR to MOC
ROC-RPW_COM-090-MDOR-PROD	Producing RPW MDOR
ROC-RPW_COM-091-MDOR-DELIV	Submitting RPW MDOR to MOC

3.3.3.6.4 Producing, deliverying and using instrument command sequences

Tab. 3.15: Instrument command sequences

Test case	Tested function
ROC-RPW_COM-100-SEQ_PROD	Producing RPW TC sequences
ROC-RPW_COM-101-SEQ_DELIV	Submitting RPW TC sequences to MOC
ROC-RPW_COM-110-SEQ_TEST	Testing TC sequences execution
ROC-RPW_COM-120-SEQ_SOUR	RPW TC sequences source and version

Note: the TC sequences execution will be tested using the MEB GSE at LESIA, the formal validation of the RPW TC sequences execution will be done by ESA with the support of the ROC team.

3.3.3.7 Instrument monitoring

3.3.3.7.1 Monitoring instrument data

Tab. 3.16: Instrument data

Test case	Tested function
ROC-RPW_MONIT-010-RPW_DATA	Monitoring RPW instrument data

3.3.3.7.2 Checking instrument command execution

Tab. 3.17: Command execution

Test case	Tested function
ROC-RPW_MONIT-020-TM_S1	Monitoring TM Service 1
ROC-RPW_MONIT-030-ISM	Checking expected instrument state
ROC-RPW_MONIT-040-SW_PATCH	Checking flight software update

3.3.3.8 Ground support

Tab. 3.18: Ground support

Test case	Tested function
ROC-GRD_SUP-010-DATA_VISU	Supporting RPW calibration campaigns on-ground: data visualization
ROC-GRD_SUP-020-DATA_DIST	Supporting RPW calibration campaigns on-ground: data distribution
ROC-GRD_SUP-030-RPW_ANOMALY	Supporting RPW anomaly investigation
ROC-GRD_SUP-040-SBM_SIMU	Supporting RPW DPU SBM1/SMB2 detection algorithms simulation

3.3.3.9 ROC infrastructure monitoring

Tab. 3.19: Infrastructure monitoring

Test case	Tested function
ROC-ROC_MONIT-010-RSS	ROC infrastructure monitoring

3.3.3.10 Communication and science support

No formal test is planned for this activity.

3.3.3.11 Non regression tests

All automatic tests are considered as non-regression tests. They will be run whenever a push on the versioning server occurs.

3.3.4 Quality control

The control and quality assurance is descrideb in the dedicated document [SPAP] (TBC). Quality reports must be provided with each test report.

4 List of TBC/TBD/TBWS

(TBW)

5 Distribution list

(TBW)

6 Appendix A - Testing guidelines

6.1 Documentation of tests

Test descriptions must have a subject and can have a body (optional) to detail the test purpose. If a test is linked to one or more specifications, the subject shall be followed by a label with the name of the specification.

Moreover the description shall respect the following rules:

- the separation between the subject, the label and the body is done with a single white line;
- the subject is limited to 50 characters;
- the subject starts with a capital letter;
- the subject line does not end with a period;
- the subject is written with the imperative mood;
- the body is wrapped at 79 characters.

6.2 Python Testing guidelines

Most of the Python tests shall be organized around Test Cases regrouping several test functions with a common setup/teardown. However, some test with no particular setup can be done using simple functions.

Each test function contains one or more expectations that test the state of the code. An expectation is an assertion that is either true or false. A test function with all true expectations is a passing test. A test function with one or more false expectations is a failing test.

Python test case classes shall be named as follow: `[FeatureBeingTested]Tests`.

Example:

```
class TvApiTests(APITestCase):
    """
    Test the RPW TM/TC Viewer (TV) API
```

(continues on next page)

(continued from previous page)

```
    :spec: [TV] My specification label

    An optional description goes here.

    """
    pass
```

Python test case methods and functions shall be named as follow: `test_[feature_being_tested]___[state_under_test]`.

Example:

```
def test_get_packet_list___no_filter(self, packet_list):
    """
    Test packet listing without filters

    An optional description goes here.

    """
    pass
```

6.3 Javascript Testing guidelines

Javascript test are organised around **Mocha** test suites

A test suite begins with a call to the global Mocha function **describe** with two parameters: a string and a function. The string is a name or title for a the test suite - the test suite shall be named as follow: `[FeatureBeingTested]Tests` . The function is a block of code that implements the suite as a succession of spec.

Specs are defined by calling the global **Mocha** function **it**, which, like **describe** takes a string and a function. The string is the title of the spec and the function is the spec, or test. A spec contains one or more expectations that test the state of the code. An expectation in **Mocha** is an assertion that is either true or false. A spec with all true expectations is a passing spec. A spec with one or more false expectations is a failing spec.

Example:

```
/**
* @test {PacketTable}
* @spec [TV] Table spec
*/
describe('PacketTableTests', function () {
  it('should have a table to list packets', function () {
```

(continues on next page)

(continued from previous page)

```
    const wrapper = shallow(<PacketTable packets={[{id: 1,
                                                    packet_name: 'TM_DPU',
                                                    packet_datetime: '2017-
↪01-01',
                                                    apid: 12,
                                                    packet_type: 'Type',
                                                    category: 'Cat'}]}/>)
    // check expected fields using assertions
    expect(wrapper.find('Table')).to.have.length(1)
  })

  it('should have a progress indicator during packets fetching', function␣
↪() {
    const wrapper = shallow(<PacketTable is_fetching={true}/>)
    expect(wrapper.find('CircularProgress')).to.have.length(1)
  })
})
```

7 Appendix B - External tools, softwares and packages

Tab. 7.1: External tools, softwares and packages

Name	Description	Reference
tox	A generic virtualenv management and test command line tool	https://pypi.python.org/pypi/tox
Pytest	A framework that makes it easy to write small unit tests as well as complex functional testing	https://pypi.python.org/pypi/pytest/3.2.3
hypothesis	A library to parametrize tests and simply generate random data matching given specifications	https://pypi.python.org/pypi/hypothesis/3.32.0
Mocha	A flexible test runner that can be used to run JavaScript tests on the server or in the browser	https://github.com/mochajs/mocha
Chai	An assertion library, similar to Node's build in assert that can be used in browser.	https://github.com/chaijs/chai
Enzyme	A JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse React Components	https://github.com/airbnb/enzyme
Jenkins	An extensible automation server used for continuous integration	https://jenkins.io/
Gitlab	A web platform used to monitorate versioning, issues and continuous integration	https://about.gitlab.com/
locust	A python utility to do distributed load testing used to perform stress tests on the MUSIC Web tools	https://pypi.python.org/pypi/locust/0.8

8 Appendix C - Beta Testing Report Template

8.1 Functional Evaluation

Tab. 8.1: Functional Evaluation

Test Case ID	Function	Test result	Comments, Ideas and Issues

[TBC]

8.2 Specific Bugs and Problems Noted

Tab. 8.2: Specific Bugs

Test Case ID	Nature of Problem	Full List of Steps to Reproduce the Problem

[TBC]

8.3 Other Generic Topics

Please comment on the following (if relevant):

- speed of user interface interactivity and of calculations
- order of screens and steps, and number of steps to complete an action
- organization of menu items
- quality of written explanations
- terms or abbreviations used

- annoying or frustrating experiences