ASTRI Mini-Array
Software Development Plan

Prepared by:	Name:	V. Conforti	Signature:	[illegible]	Date:	Nov 5, 2020
Verified by:	Name:	G. Tosti	Signature:	[illegible]	Date:	Nov 5, 2020
Approved by:	Name:	G. Tosti	Signature:	[illegible]	Date:	Nov 5, 2020
Released by:	Name:	S. Scuderi	Signature:	[illegible]	Date:	Nov 5, 2020

Main Authors:
Contributor Authors:

Table of Contents

INDEX OF FIGURES & TABLES

Document History		
Version	***Date***	***Modification***
1.0	Nov 5, 2020	First version after internal check of the ASTRI-MA software team

1. Introduction

The **ASTRI Mini-Array (MA)** is an INAF project aimed to observe astronomical sources emitting at very high-energy in the TeV spectral band. The ASTRI MA consists of an array of nine innovative Imaging Atmospheric Cherenkov telescopes that are an evolution of the two-mirror ASTRI Horn telescope successfully tested since 2014 at the Serra La Nave Astronomical Station of the INAF System of Catania. Each telescope will be equipped with the new version of the ASTRICAM Silicon photomultiplier Cherenkov Camera. The main science goals of the ASTRI MA encompass both galactic and extragalactic science. The nine telescopes will be installed at the Teide Astronomical System, operated by the Instituto de Astrofisica de Canarias (IAC), on Mount Teide (~2400 m a.s.l.) in Tenerife (Canary Islands, Spain). The ASTRI MA will be operated by INAF on the basis of a host agreement with IAC.

1.1. Purpose

The ASTRI software development is managed in an open and transparent way. The community involves mainly two subjects:

- the INAF team (which includes also other public research institutions such as the Università of Perugia and INFN);
- the private companies.

This document implements a tailoring of the template in Annex O of RD1. In particular it applies only to section 5 of the template because the prime sections concern management aspects detailed in AD1.

1.2. Scope

Software is found at all levels, ranging from system functions down to firmware, including safety and mission critical functions. Therefore, a special emphasis is put on the system-software relationship [AD1] and on the verification and validation of software items according to the Software Quality Plan [AD5]. The scope of this document is to provide guidelines for development, operation, maintenance, verification and validation activities.

1.3. Contents

1.4. Definitions and Conventions

1.5 Abbreviations

2. Applicable and reference documents

No text here. Put text in the subchapters

2.1. Applicable documents

[AD1] ASTRI MA Software Engineering Management Plan
[AD2] ASTRI MA Glossary and Abbreviations, ASTRI-INAF-LIS-2100-001
[AD2] ASTRI MA Top Level Use Cases, ASTRI-INAF-SPE-2100-001
[AD3] ASTRI MA Top Level Software Architecture, ASTRI-INAF-DES-2100-001
[AD4] ASTRI MA Science Requirements
[AD5] ASTRI MA Software Assurance Plan
[AD6] ASTRI MA System Engineering Management Plan ASTRI-INAF-PLA-2000-001
[AD7] ASTRI MA management plan
[AD8] Interface Management Plan

2.2. Reference documents

[RD1] ECSS-E-ST-40-C "European Cooperation for space standardization - Space engineering software" - ESA ESTEC, 6 March 2009.

3. Software development and validation approach

In accordance with [AD1], we apply the principle of 'customer–supplier' relationship, assumed for all developments of the ASTRI MA software. The customer is, in our case, the procurer of the software for a system, subsystem, set, equipment or assembly. The concept of the "customer–supplier" relationship is applied recursively, i.e. the customer can be a supplier to a higher level (i.e. system level).

According to the WBS detailed in AD7, scope of this document are the following relationships:
- System Engineer (customer) - System Software Engineer (supplier);
- System Software Engineer (customer) - Software Coordinator (supplier);
- Software Coordinator (customer) - Software package coordinator (supplier);

The ASTRI organization identifies the system engineering team, the system software engineering team, and the software coordinator [AD1]. Five major high-level sub-packages have been identified within the ASTRI MA software system:

- SCADA;
- Archive;
- Data Processing;
- Science Support System;
- Simulations.

The coordinators of the packages above apply the customer-supplier relationship recursively.

The software coordinator with the support of the system software engineering team is responsible to coordinate the overall SW activities, related to the ASTRI mini-array. He reports directly to the ASTRI software engineer for the overall program and interfaces the SW PA. Main tasks to be accomplished are:
- keep track of the progress of the activities and report relevant issues to the higher level;
- ensure proper follow-on of the ASTRI SW overall and coordinate the activity performed by the WP;
- ensure proper consistency between the various software components;
- support other major system activities having direct link with the SW (operations definitions, DB activities, AIV plans);
- contribute to finalize the SW documents to be prepared.

The suppliers within the software packages are intended either private contractors or development teams within the ASTRI organization.

The definition of the interfaces between hardware and software items follows the Interface Management Plan [AD8]. Software items are defined in the software PBS at different levels.

Reviews are the main interaction points between the customer and the supplier. They also synchronize software engineering processes. The reviews relevant to the software engineering processes are the Preliminary Design Review (PDR), Critical Design Review (CDR), Acceptance Test (AR), and Operational Readiness Review (ORR). The reviews occur at different levels in the customer–supplier hierarchy and are sequenced according to the overall system level planning.

3.1 Strategy of the software development and validation

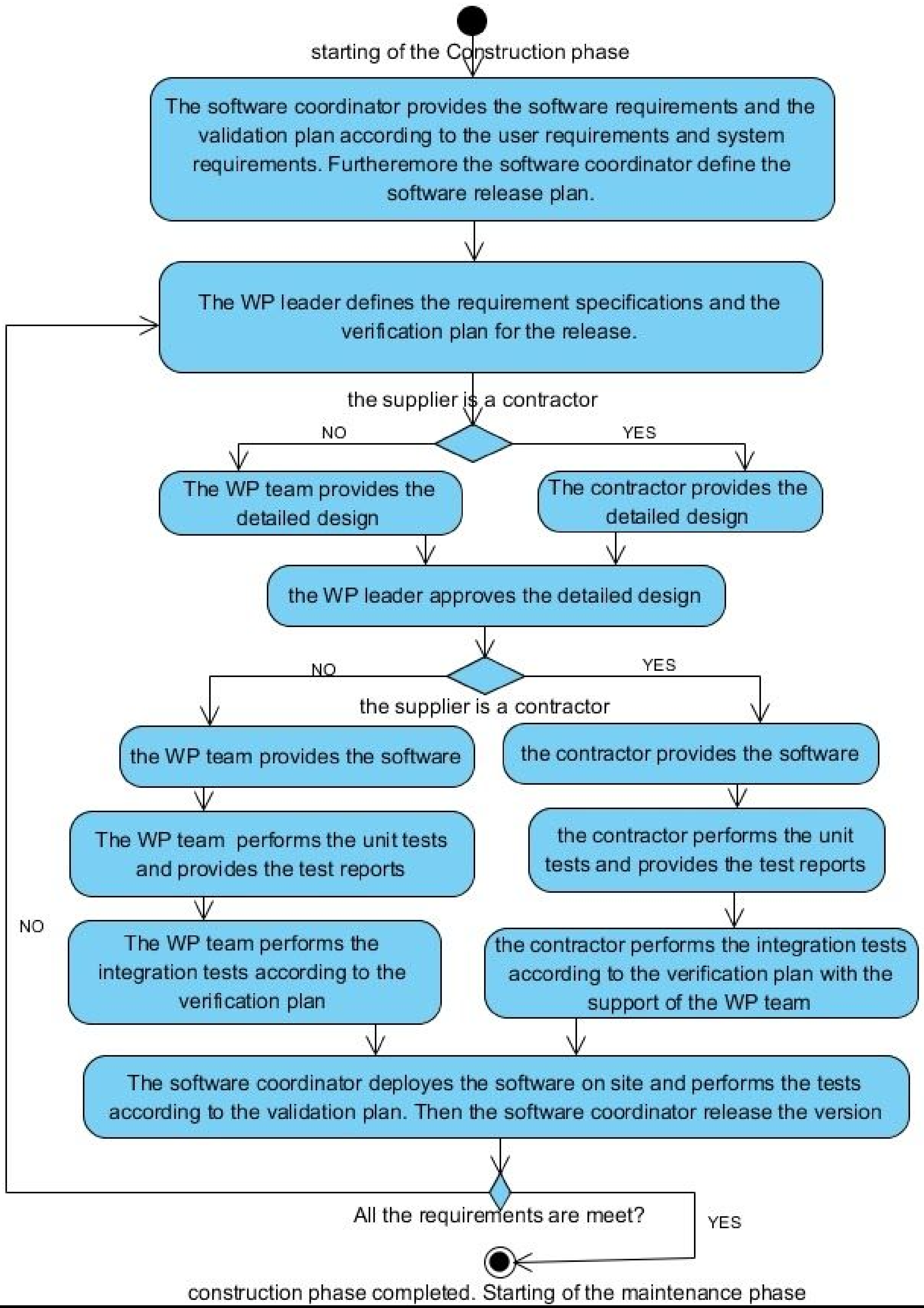
This section describes the overall strategy to the software development through the UML activity diagram below.

The typical waterfall approach will be adapted in accordance to the ASTRI system engineering approach [AD6] in order to introduce an iterative-incremental development method in which several intermediate versions will be verified, validated and delivered before starting the validation of the final SW version.
Each intermediate SW version will be delivered, integrated, verified and validated with respect to the requirements baseline and the time schedule which correspond to the implemented functionalities.

The Validation approach is supporting the concept of an incremental verification and validation performed on different test benches and at different levels, both on the contractor test facility and the INAF test facilities. The final SW validation will be performed at the two ASTRI MA main sites: the Array Observing Site (AOS) at Teide, and the ASTRI MA Data Center in Rome.

The approach is that each SW component will be verified and integrated by the INAF sub-package coordinator with the support of the responsible contractor and/or the INAF development team. The intermediate/final software sub-package validation will be in charge of the Software coordinator with the support of the WP coordinators.

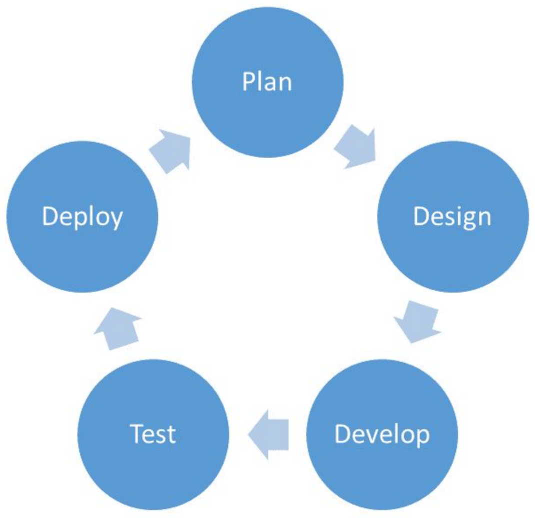
During the development process after the PDR, any change to the requirements baseline, the technical specifications, the architecture and detailed design shall follow a formal Request For Change procedure.
The AD1 details the project phases and the reviews which shall be executed.


starting of the Construction phase
The software coordinator provides the software requirements and the validation plan according to the user requirements and system requirements. Furtheremore the software coordinator define the software release plan.
The WP leader defines the requirement specifications and the verification plan for the release.
the supplier is a contractor
NO
YES
The WP team provides the detailed design
The contractor provides the detailed design
the WP leader approves the detailed design
NO
YES
the supplier is a contractor
the WP team provides the software
the contractor provides the software
The WP team performs the unit tests and provides the test reports
the contractor performs the unit tests and provides the test reports
NO
The WP team performs the integration tests according to the verification plan
the contractor performs the integration tests according to the verification plan with the support of the WP team
The software coordinator deployes the software on site and performs the tests according to the validation plan. Then the software coordinator release the version
All the requirements are meet?
YES
construction phase completed. Starting of the maintenance phase

3.2 Software project development lifecycle

3.2.1 Software development life cycle identification

In software engineering, a software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle (SDLC). The activities can be broken down into five core categories: plan, design, develop, test and deploy. Below is a graphic representation which displays a typical software development life cycle.



Plan: the planning stage concerns the feasibility study in financial, operational and technical areas. The main input is the science requirements document [AD4]. The deliverables at this stage are the use cases document [AD2] and the software architecture document [AD3].

Design: the design stage starts with a clear definition and approved expected scenarios detailed through use cases. The deliverables at this stage are each subsystem: the Interface Control Documents (ICD), the Software Design Documents (SDD), the Software Requirements Specification (SRS) , the Software Verification Plan (SVerP) and the Software Validation Plan (SValP).

Develop: after the most appropriate design has been selected, implementation starts immediately. Prototyping activities are taken into account. Programmers should develop the software according to the SDD and at the same time follow the coding standards and development rules defined in this document. The deliverables at this stage are the code properly versioned in the software repository.

Test: Software testing should be conducted at all stages as a sub-stage. Nonetheless, two major ones should be done by programmers, end-users and quality assurance experts. The reason is that programmers know the best of how the program works and therefore they can identify the most vulnerable areas of the software. End-users would pay more attention to their routine tasks which can help to ensure the software can fulfill the requirements. Last but not least, quality control experts examine the software as whole from various perspectives such as architecture, security, integration with other systems. As a result, a few different types of test plans should be prepared for the three groups of testers to conduct at the test stage. The deliverables at this stage are for the system as a whole and for each subsystem the test reports and the Integration test report according to the SVerP.

Deploy: First thing to do at deployment stage is to verify all the test cases were run to ensure successful software execution, comprehensiveness and correctness. Final decision should be made if the software should be released and deployed to the production environment. Guideline documentations should also be prepared such as Installation guide, administration guide and user guide. The deliverables at this stage are the test reports according to the validation activities detailed in SVaIP, the Software Release Document (SRD) and the Software User Manuals (SUM).

3.2.2 Relationship with the system development life cycle

This section describes the phasing of the software life cycle to the system development life cycle and it will be detailed according to the System Engineering team.
This section is TBD.

3.2.3 Reviews and milestones identification and associated documentation

The table below resumes the main reviews, the expected deliverables and the rules of customer and supplier.

REVIEW	DELIVERABLES	RULES
Concept Review	● Use Cases Document ● Logical Software Architecture ● System Requirements	The main actor in this phase is the customer who defines what shall be realized by the user point of view.
Preliminary Design Review (PDR)	● Software Requirement Specification (SRS) ● Use Case Document (UCD) / User stories ● Interface Control Document (ICD) ● preliminary Software Verification Plan (SVerP) ● preliminary Software Validation Plan (SValP) ● preliminary Software Design Document	The main actor in this phase is the supplier who defines what shall be realized by the developer point of view. In addition the supplier also provides the solution which includes how shall be developed, verified and validated the software. The customer approves the deliverables.

	(SDD)	
Acceptance Test	● test reports (verification) ● preliminary user manual	The main actor is the supplier who has in charge the verification of the product according to the requirement specification with relative test reports. The customer controls and assesses the results.
Operational Readiness Review	test reports (validation) user manuals	The main actor is the customer who has in charge the validation of the product according to the user requirements.

3.3 Software engineering standards and techniques

3.3.1 Software requirements analysis methods

Requirements will be documented and traced using the Enterprise Architect tool.

3.3.2 Design methods and standards

The Unified Modelling Language (UML) will be used to design the SW. UML focuses on a standard modelling language, not a standard process. The structural elements helping to build models and design in UML can be i.e. diagrams to illustrate the system (sequence and collaboration diagrams), model elements to represent object oriented concepts (classes, object, messages and relationship). These UML elements can be classified in the following components:

1. Views: show different aspects of the modeled system. A view can be considered an abstraction consisting in a number of diagrams containing information that put in evidence a particular aspect.
2. Diagrams: are graphs that describe the content in a view showing model element symbols opportunely arranged to illustrate a particular perspective of the system under analysis or development. (e.g. use case diagrams, sequence diagrams, collaboration diagrams, etc).
3. Model elements: concepts used in diagrams are called model elements and represent common object oriented concepts such as classes, objects, messages and relationships. A model element has a corresponding view element which is the graphical representation of the element or the graphical symbol used to represent the element in a diagram.
4. General mechanism: allows adding additional information into a diagram that cannot typically be represented using the basic abilities of the model elements (adornments, notes, specifications).

The diagrams will be produced using the Enterprise Architect tool.

3.3.3 Programming language

C++, Python and Java are the programming languages to be used under the Alma Common Software (ACS) framework.
C language is proposed as a programming language to be used to develop other applications.
Any other programming language shall be evaluated and approved by the customer.

3.4 software development and software testing environment

3.4.1 Software development environment

The software development environment is used during the design, coding and unit testing of the WP components. It has to be available and maintained during the whole operational life of the software components to allow their proper SW maintenance.
The selection of the software development environment items has to pursue the objective to maximize the commonality in terms of tools and methodologies to be used by the different SW Contractors; however the following aspects have to be taken into account:

- The background of the SW Teams involved in the SW development and the usual standards adopted in their Companies. For example, some teams could be used to program in different languages whereas others have always worked with C. In this case forcing a programming language, and so also all related development tools, could not be beneficial for the Project since the efficiency of the Team would be degraded and the quality of the SW product would be put at risk.
- The availability of existing SW frameworks and/or libraries. If SW reuse is foreseen then the relevant software development environment has necessarily to be used. The reuse of existing software must be discussed in the SDD.
- The specific features of the SW to be developed, or the relevant HW target platform, could become driving factors for the selection of some software development environment items.

According to the above, the following table summarizes the proposed software development environment tools. They will be confirmed case-by-case for each specific SW product to be developed by the relevant Contractor.

item	proposed tool	Remarks
Requirement management	Enterprise Architect	
Design tool	Enterprise Architect	Based on UML
Development environment	Eclipse, Visual Studio, Netbeans	Any equivalent Tool, to be agreed with the Contractor
Compiler	Java 8, Python2 / 3, C++, GCC	Any equivalent Tool, to be agreed with the Contractor
Operating System	Centos 8	

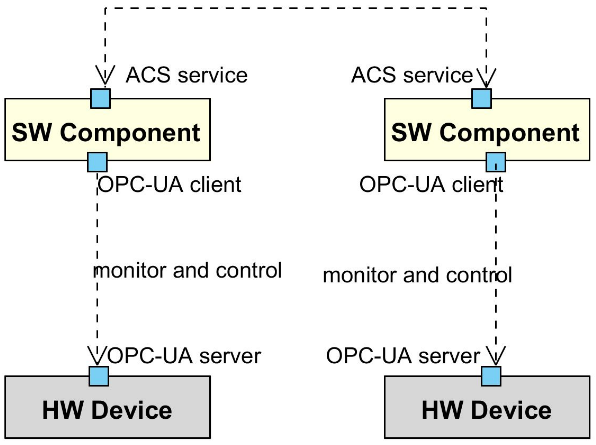
Development Framework / Middleware	ACS	
Engineering DB	TBD	
DB	TBD	
configuration management control	TBD	
produce development oriented documentation	doxygen, MS Office, Latex, Google documents	
Software version control	GitLab	
development virtual machine	Virtual machine, containers (docker, singularity)	prepared by ASTRI - ICT
integration environment	test bed	prepared by ASTRI - ICT
project management	redmine	
document management	redmine	dmsf plugin
TBD		

The SW release to be used, for each tool, will be jointly agreed and frozen with the SW contractors during the project.

3.4.2 Software Test environments

3.4.2.1 Software integration test model

The Software test environments can be classified into two distinct categories, depending on the presence (or not) of some representative “HW in the loop”. In information technology, the systems integration is the process of linking together different computing systems and software applications physically or functionally, to act as a coordinated whole. The ASTRI Collaboration selected the OPC-UA (OPC - Unified Architecture) to connect the software to the hardware devices, and ACS (ALMA Common Software) to interface the software components to each other within the application layer as depicted below:



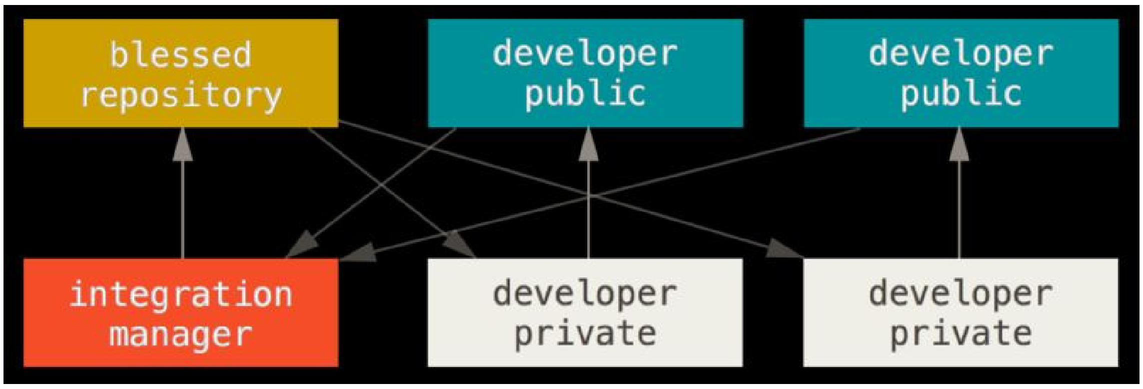
We plan three levels of integration:

- Lowest level: the goal is to test the interfaces between each hardware device and its related software. The software shall be able to monitor and control all the hardware device properties.
- Middle level: the goal is to test the interface between a software component and its neighbours, both the incoming and out coming calls.
- Top level: the goal is to test the system as a whole by executing the scenarios detailed in the use cases.

We provides for the integration activities the following tools:

- Git repo for the version control;
- Jenkins to perform the continuous integration tests;
- Test report and release templates to document the software tests and the releases;

The Git repo shall be used for the software versioning at any level (firmware, prototyping software, production software, third-party software). The Git tag system shall be used to set a component version or in general a software version for a specific physical machine. Jenkins exploits the setup and unit test to execute the continuous integration.
The virtual machine and container system shall be used to execute the middle level integration. For instance the docker container system provides a light-weight system to create the machine images to simulate and test interaction among software elements hosted on different machines. The virtual machines provide an environment similar to the real system in order to facilitate local tests. The test bed provides a virtual environment in which all the production machines are simulated. The ASTRI software integration follows the Integration-Manager workflow depicted in the following picture:



The models foresee that the developer performs the code versioning in the developer private repo (in Sandbox). When the software version is ready for the test the developer pushes the code in the dev branch of component repo.

The integration manager pulls the software of the repo which shall be tested and perform the test properly. When the software is ready for the release then all the components are pushed on the blessed repo (master branch).

3.4.4.2 Software integration test note

The integration activities require specific development rules to support the integration processes.

1. The software components shall be tested (unit) and documented (with user manual and setup instructions); The first step shall be always to produce a simulator in order to do not lock the other developer jobs;
2. The developer shall use the virtual machine for the development which provides a standardized environment, very similar to what will be used in operations. OPC-UA is used throughout to integrate the software with the hardware devices, and ACS to integrate higher-level software components with each other;
3. A Git repository shall be used for the releasing of software. The developer shall use the repository for the intermediate versions during the software development;
4. It is mandatory to perform the lowest level integration (for components which interface hardware devices) and middle level integration (exploiting either virtual machine or test bed) before any release. In case of success the developer shall tag the version;
5. It is mandatory to perform the top level integration (exploiting the test bed) before any major release of the software, and before the deployment in the production environment. The software on-site can be updated and a new version of the Software Release Document (SRD) shall be published;
6. Continuous integration, consisting of a full build and execution of all unit tests, is done via Jenkins. The developer is in charge of producing and maintaining the job jenkins properly.

3.4.4.3 Software Testing facilities

- jenkins for the continuous integration;

- test bed for the software integration tests;
- ASTRI-Horn Telescope Prototype;
- TBD

Jenkins is a software platform that supports the continuous integration tests. The WP leader is responsible for implementing the corresponding job.

The test bed is a set of virtual machines installed at IASF BO which reproduce the same real machine on site. The virtual machines in the test bed have the same configuration (networking, users, operating system, dns..) of real machines in order to allow the tester to perform very effective tests.
All the machines in the testbed as the machine on-site have the ASTRI Git user read only in order to allow the integrator to download or update the software.

ASTRI-Horn is the ASTRI telescope prototype installed at SLN and can be used to perform some tests according to the scientific activities scheduled for ASTRI-Horn and the guidelines and constraints which will be provided by the ASTRI-Horn responsible.

3.4.4.4 Note for contractors

Note: the selected contractors will use their own tools. Nevertheless the product owner shall define in the contract the following items:
- the contractor shall release periodically the software according to the milestone. The product owner shall define the multiple milestones in order to provide functionality for each release and then to benefit the test activities.
- the contractor shall use the git environment provided by INAF for their software release;
- the contractor shall use the virtual machine provided by INAF to test their software before the release;

3.5 Software documentation/deliverables plan

The documentation production, writing and release shall follow the rules and guidelines provided by the ASTRI organization. [TBD]

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the life cycle. The table belows resume the deliverables and the specifications to produce or update.

CYCLE	DELIVERABLES	Responsible
Plan	● Use Cases Document ● Logical Software Architecture ● System Requirements	Customer.

Design	• Software Requirement Specification (SRS) ○ prepared by supplier and accepted by customer; • Interface Control Documents (ICD) • Software Design Documents (SDD) • Software Verification Plan (SVerP) ○ prepared by supplier and accepted by customer. It shall comply with the SRS. • Software Validation Plan (SValP) ○ prepared by the customer. It shall comply with the System requirements.	Supplier with the supervision of the customer.
Develop	• Code of the software under version control • job jenkins • sonarQube	Supplier.
Test	• Test Reports • Integration Test Report.	Supplier with the supervision of the customer.
Deploy	• Software Release Document (SRD) • Software User Manuals (SUM)	Customer with the support of the supplier.

3.6 Operational support and SW Maintenance

TBD

3.7 Software licensing

TBD