EOD Earth observation data processing preparation and evaluation

25/01/2016

dEcentralized repositories for traNsparent and efficient vIrtual machine opErations

EOD Earth observation data processing preparation and evaluation

Responsible author(s): **Jonathan Becedas**

Co-author(s): Jose Julio Ramos

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 644179.

Revision history

Administration and summary		
Project acronym: ENTICE		
Document identifier:	ENTICE D6.3	
Leading partner: Deimos		
Report version: 1.0		
Report preparation date: 20.12.2015		
Classification: PP (Restricted to other programme participants)		
Nature: Other **(**Internal Document)		
Author(s) and contributors: Jonathan Becedasand Jose Julio Ramos		
Status:	-	Plan
	-	Draft
	-	Working
	X	Final
	-	Submitted
	-	Approved

The ENTICE Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

Date	Edited by	Status	Changes made
19.11.2015	Jonathan Becedas	Draft	Initial outline
18.12.2015	Jonathan Becedas	Draft	Version for review
18.12.2015	Attila Kertesz	Working	Review comments and suggestions
23.12.2015	Nishant Saurabh	Working	Review Comments and suggestions
18.01.2016	Jonathan Becedas	Working	New version considering A. K. review.
25.01.2016	Jonathan Becedas	Final	New version considering N. S. Review.

Notice that other documents may supersede this document. A list of latest *public* ENTICE deliverables can be found at the ENTICE Web page at http://www.entice-project.eu/.

Copyright

Citation

J. Becedas and J.J Ramos. (2015). EOD Earth observation data processing preparation and evaluation. ENTICE Consortium, c/o Elecnor Deimos, http://www.entice-project.eu.

Acknowledgements

The work presented in this document has been conducted in the context of the EU Horizon 2020. ENTICE is a 36-month project that started on February 1st, 2015 and is funded by the European Commission.

The partners in the project are UNIVERSITAET INNSBRUCK (UBIK), MAGYAR TUDOMANYOS AKADEMIA SZAMITASTECHNIKAI ES AUTOMATIZALASI KUTATOINTEZET (SZTAKI), UNIVERZA V LJUBLJANI (UL), Flexiant Limited (Flexiant Limited), WELLNESS TELECOM SL (WTELECOM) and Deimos CASTILLA LA MANCHA SL (Elecnor Deimos Satellite Systems). The content of this document is the result of extensive discussions within the ENTICE© Consortium as a whole.

More information

Public ENTICE reports and other information pertaining to the project are available through ENTICE public Web site under http://www.entice-project.eu.

List of figures

List of tables

1 Introduction

This document presents the validation plan of the Earth observation data processing pilot. Hereby the process that shall be carried out by DEIMOS to perform the testing of the software is described; the requirements and objectives to deploy and validate the EOD use case are summarized. The objective is to provide adequate confidence for development, testing and validation campaigns.

This document shortly describes the four use case scenarios to be tested for the validation of the EOD pilot together with the description of the system architecture (Section 2); the virtual machines required to implement the system and the requirements to deploy the EOD pilot are also described. A brief introduction of the requirements to validate the EOD pilot is done in Section 3. These requirements were detailed described in "Deliverable D.2.1. Requirements specification for ENTICE". In addition, the Validation Plan for the EOD pilot, including the software validation process, the validation approach, the software validation test facilities and the software validation test reporting (Section 4). Furthermore, the general and the verification and validation administrative procedures are specified in Section 5. Finally Section 6 describes the software tests to be executed to validate the system at integration, system and acceptance level.

2 Virtual Machines description

2.1 System Architecture

The distributed architecture in cloud is virtualized and constituted by: a GS Monitor, which continuously pools over the different ground stations subscribed to the system; a Product Processors module, which processes the raw data obtained from the satellite; an Archive and Catalogue module, which stores and classifies the processed images and module providing User Services – as shown in Figure 1.

The architecture is implemented with the ENTICE middleware through functional descriptors. These functional descriptors define the system. ENTICE analyses, synthesizes and optimizes the VMIs by automatically removing unnecessary VM content like unused libraries or old log files or merging different VM templates stored in the system. Figure 1 shows a scheme of the implementation concept with ENTICE.

During the trialling of the distributed data centre proposed the following proposed scenarios will be validated:

• **Scenario 1. "Nominal Operation"** – It provides the real performance metrics and present real system requirements for our operational system of one satellite. This scenario will be the metrics baseline.

• **Scenario 2a. "Land Management" -** Governments, planning offices and agricultural agencies need to characterize landscape and crops in vast rural regions. They demand a multi-temporal product which allows analysing the land cover dynamics and detecting changes during a vast amount of time. The main objective is to optimize the archive of satellite recordings in order to facilitate reprocessing campaigns and time-series analysis.

• **Scenario 2b. "Infrastructure monitoring" –** Infrastructure monitoring is usually done during the construction or operations of major infrastructures in remote, hazardous areas such as high performance railway lines in the desert. These infrastructures would need constant monitoring to minimize the impact of the ambient conditions. The main objective is to optimize the processing of the satellite imagery to facilitate on-demand processing.

• **Scenario 2c. "Disaster response" -** Upon the event of a disaster, say an earthquake on a remote area, government and safety agencies would need to acquire, as soon as possible, images from the affected area before, during and after the event. Those images would be later demanded by a huge number of users from all around the world, either to manage emergency services and to assess the damages. The objective is to optimize the distribution of satellite imagery at a global scale by creating a Content Distribution Network.

More detailed information about the architecture and the use cases can be found in *Deliverable 2.1 Requirements specification for the ENTICE environment*.

With the EOD implementation based on ENTICE, it is expected to reduce the VMI size by 60%, the VMI delivery time by 30%, the deployment time by 25%, the VM storage by 25% and at least 25% VM cost reduction in the use of cloud infrastructures.

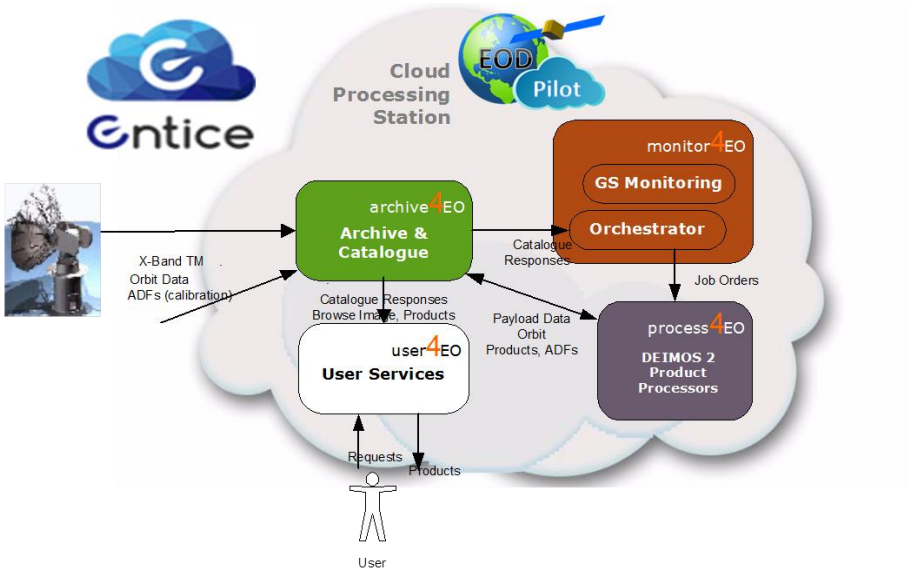

Figure 1. EOD System Architecture

2.2 Virtual Machines Description and requirements to deploy the EOD pilot

THE EOD pilot will be implemented in cloud using a distributed architecture. Each of the modules of the architecture will be implemented in one separate VM instance. This facilitates the incorporation of elasticity feature to the system, and auto-scaling can be done in function of the demand of processed images by the users. The following VMs will be used to implement the architecture and run the different scenarios described in deliverable *D2.1 Requirements specification for ENTICE*: Product Processor, Archive, Catalogue, Orchestrator, GS Monitor and User Services.

Table 1 Product Processor VM specification

PRODUCT PROCESSOR		
VM runtime		
CPU Clock	2.53GHz	
CPU Count	4	
RAM	16GB	
HDD	20GB+100GB	
Network	Private	
Constrains		
VM	SHARED STORAGE and the other PRODUCT PROCESSOR VMs	Near
	ARCHIVE AND CATALOGUE	Geography redundant
Disk	SSD	Type

Analytics	
Expected Users	5-10

Table 2 Archive VM specification

ARCHIVE		
VM runtime		
CPU Clock	2.53GHz	
CPU Count	4	
RAM	16GB	
HDD	20GB+500GB	
Network	Private	
Constrains		
VM	CATALOGUE	Near
	N/A	Geography redundant
Disk	SAS	Type
Analytics		
Expected Users	5-10	

Table 3 Catalogue VM specification

CATALOGUE		
VM runtime		
CPU Clock	2.53GHz	
CPU Count	4	
RAM	4GB	
HDD	20GB	
Network	DMZ	
Constrains		
VM	ARCHIVE	Near
	ARCHIVE	Geography redundant
Disk	SAS	Type
Analytics		
Expected Users	5-10	

Table 4 Orchestrator VM specification

ORCHESTRATOR	
VM runtime	
CPU Clock	2.53GHz
CPU Count	4
RAM	4GB

HDD	20GB	
Network	Private	
Constrains		
VM	GS MONITOR, ARCHIVE AND CATALOGUE, PRODUCT PROCESSORS	Near
	ARCHIVE, CATALOGUE, USER SERVICES	Geography redundant
Disk	SAS	Type
Analytics		
Expected Users	5-10	

Table 5 GS Monitor VM specification

GS MONITOR		
VM runtime		
CPU Clock	2.53GHz	
CPU Count	1	
RAM	2GB	
HDD	20GB	
Network	Private	
Constrains		
VM	N/A	Near
	N/A	Geography redundant
Disk	SAS	Type
Analytics		
Expected Users	5-10	

Table 6 User Services VM specification

USER SERVICES		
VM runtime		
CPU Clock	2.53GHz	
CPU Count	4	
RAM	8GB	
HDD	50GB	
Network	DMZ	
Constrains		
VM	ARCHIVE AND CATALOGUE, SHARED STORAGE	Near
	ARCHIVE AND	Geography redundant

	CATALOGUE, USER SERVICES	
Disk	SAS	Type
Analytics		
Expected Users	5-10	

The number of VMs changes in function of the scenario to be tested. Summarizing, the following tables (Table 7 to Table 10) show the required instances.

Table 7: System configuration for Scenario #1

	Product processors	Archive	Catalogue	Monitor	User Services	Orchestrator
SC1	7	2	1	2	1	1

Table 8: System configuration for Scenario #2a

	Product processors	Archive	Catalogue	Monitor	User Services	Orchestrator
SC2a	7	20	5	2	1	1

Table 9: System configuration for Scenario #2b

	Product processors	Archive	Catalogue	Monitor	User Services	Orchestrator
SC2b	21	2	1	2	1	1

Table 10: System configuration for Scenario #2c

	Product processors	Archive	Catalogue	Monitor	User Services	Orchestrator
SC2c	7	2	1	2	10	1

3 Requirements to validate the EOD use case

The functional, non-functional and interface requirements to deploy the EOD use case are described in detail in *D2.1 Requirements specification for ENTICE.* In that document, the functional requirements are divided into *behaviours, capabilities, knowledge model, VMI images* and *evaluation metrics;* the non-functional requirements are divided into *performance, storage, cost model, technology, privacy, compliance, security, reliability, availability, maintainability and*

portability; and the interface requirements are divided into *hardware, software* and *communications.*

4 Validation plan

This section defines the validation plan to follow in the development of the EOD pilot use case.

4.1 Software Validation Process Overview

4.1.1 Organization

This subsection describes the organization of the Software Testing Activities.

4.1.1.1 Roles

There are 4 project team roles involved in the Software Test Plan: Principal Investigator (PI), Software Product Assurance Engineer (SPAE), Technical Manager (TM), and Project Engineer (PE). The PE will create/execute the tests and fix the defects found during the tests.

4.1.1.2 Reporting Channels

During the execution of the Software Testing Campaign, any errors found by the tester shall be reported to TM.

4.1.2 Levels of authority for resolving problems

The Project Engineer shall run the Software Testing Campaign to verify the SW implementation and integration.

After or even during the run of the Software Testing Campaign, the Project Engineer shall monitor the issues to check if any problem was found. Whenever a new problem is found, the developer shall fix it and update its status.

The Software Product Assurance team shall monitor the execution of the SW testing Campaign, ensuring that all defects found are reported and fixed. New instances of the Testing Campaign may be required to ensure that all defects are fixed. Defects found shall be reported in the Software Product Assurance Report.

4.1.3 Relationships to other activities

The result of the execution of the Software Testing Campaign shall be added to the Software Verification Report or similar documentation.

4.1.4 Master Schedule

The Software Testing Activities will be performed in accordance to the Software Development defined in the ENTICE proposal and will fulfil the planning shown in Figure 2. “Task 6.3

Integration of ENTICE and pilot use case" will start on 29th January 2016; the integration of the preliminary EOD pilot components will take place. A set of integration tests will be done to validate the functioning of the EOD components working together. In parallel the EOD pilot components will be further developed from the preliminary approach, starting with Task 6.2 on 29 January 2016 to provide a first prototype on 1st July 2016. This milestone coincides with the delivery of a first prototype of the ENTICE environment. A first batch of system tests will be performed to validate the performance of the EOD prototype.

In August 2016 the validation of the ENTICE environment will start with Task 2.3, providing the results on 31st October. During this stage the EOD prototype will be further developed, and the EOD pilot will start its deployment in the ENTICE environment for its integral use. ENTICE evaluation will finish on 31st October, and the environment will be further developed and tuned by considering the results of such an evaluation. In parallel, the EOD pilot system will be further developed and integrated with ENTICE. EOD integration and system tests will be carried out to provide the final version of the pilot use case on 29th September 2017 and the final ENTICE system on 31st January 2018.

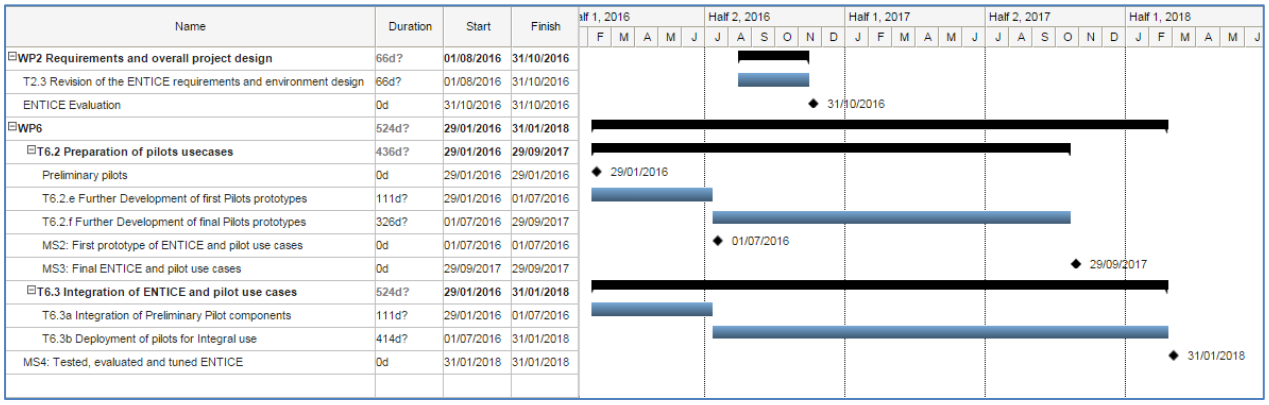

Figure 2. EOD Pilot Software development validation activities planning

Verification and validation activities of the EOD pilot are immersed in WP 6, and specifically in Task 6.2. Table 11 summarizes this relation.

Table 11 Relation of the EOD pilot verification and validation activities with the project

Work Package	Relation of the EOD pilot verification and validation activities with the project
WP 6	**Task 6.2. (Month 4-32): Preparation of pilot use cases:** • Preparation of the EOD pilot use case with the objective of testing and evaluating the ENTICE environment. • To specifically develop the EOD pilot use case. • To tune the EOD pilot use case for the evaluation of the ENTICE environment.

Task 6.3. (Month 13-36): Integration of ENTICE environment and pilot use cases:

- Integration of the EOD pilot use case and the ENTICE environment.
- Deployment of the EOD pilot use case for its integral use.

Task 6.4. (Month 13-36): Testing, evaluation and tuning of ENTICE environment with the pilot use cases

- Test, evaluation and tuning of the EOD pilot use case according to the project objectives.

4.1.5 Resource Summary

The Software Verification and Validation activities shall be performed by the development team, under the Principal Investigator supervision, using the following hardware and software tools:

- Junit - Unit testing framework
- Enterprise Architect - Design tool for keeping traceability with the specifications.

4.2 Validation Approach

This section describes how the software will be validated, and identifies the required resources and inputs, expected outputs and criteria for suspending and resuming the validation if a problem is encountered. If there is more than one type of validation task, e.g. different software items are validated using a different verification method.

Each requirement in *Deliverable 2.1 Requirements specification for the ENTICE environment*, section 2.12 shall be assigned a verification method. This verification method can be one or a subset of the following:

- **Test:** execute a software component or system. Tests shall be the primary verification method.
- **Inspection:** a visual check of the tested component.
- **Review:** review of documentation.
- **Analysis:** this method shall be applied after discarding the feasibility of test, inspection or review methods. It is often based on the simulation of a functional unit to be verified, which implies software modelling. The results of this simulation is then analysed by the engineering disciplines and an acceptable level of confidence is reached to accept it via analysis reports for the close-out of the requirements. In general, the analysis method is an “ad hoc” assessment of the software.

All those methods shall meet the objective of demonstrating that the requirements are fulfilled. As stated above, we will refer to Inspection, Review and Analysis as supplementary validation methods.

4.2.1 Features to be tested

It shall be verified that the tested operations conform to their expected behaviour, i.e., that each operation generates the expected outputs for different sets of inputs. During the different SW layers integration, the high-level testing will be based in the change of the stub for the real software.

4.2.2 Features not to be tested

The functionality of the operations that could require some hardware behaviour will be not tested, as a result of which no hardware simulation is foreseen.

4.2.3 Test Pass/Fail Criteria

Upon drive/stubbing, each test case will have a point of decision for assessing if the test passed or failed, based upon an "AND" of the following list of criteria (when applicable):

- **SW Under Test (SUT) return value:** this will be checked by the SUT wrapper function that is called by the test driver, and is responsible for calling the main function, therefore verifying its return value after it exits. If this value is different than the expected one, it will return a "FAIL" result to the test driver;
- **SW Under Test global variables' values after return:** this will also be checked by the SUT wrapper function, which shall verify the expected values of any global variables set by the SUT, after it exits. If the value of any of the global variables is different than the expected one, it will pass a "FAIL" result to the test driver;
- **Values of parameters are passed as arguments to other functions called by SUT as a consequence of the test-driven execution path:** these will be checked by the functions called, which will be stubbed to make this verifications. If any of these stubbed functions is not called as expected by the SUT (the adequate number of times and with the expected parameters), they will set the result variable (to be returned to the test driver by the wrapper function) to a "FAIL" result.

4.2.4 Suspension Criteria and Resumption Requirements

Tests will be run sequentially and every test execution will be documented in a separate test report.

For problem reporting and corrective actions, the standard procedure will be followed, making use of software problem reporting, non-conformance reporting or software modification reporting.

When a test fails, an appropriate problem report will be generated and sent to the appropriate parties for actioning.

If a given test is affected by an open software problem, the execution of that test will be suspended pending a correction to the software problem. It will be resumed when an

appropriate correction is made, at which time the test will be repeated. If the SPR (Software Problem Report) identifies other tests that may be affected by the problem, they will also be repeated (regression testing).

Overall testing may be suspended if a serious error or deficiency is discovered in the test environment or one that affects the whole or a large part of the integrated software under test.

4.2.5 Manually and Automatically Generated Code

The software does not have any automatically generated source code from a model. Thus all the testing activities aim manually generated code.

4.3 Software Validation Test Facilities

The validation activities include on on-site Factory Acceptance Test (in DEIMOS premises) which will be performed by the development team, under the Principal Investigator supervision, and with the presence of the customer using the same hardware platform used for the development.

4.4 Software Validation Test Reporting

The results of the validation tasks (tests, inspections, analyses or reviews) will be reported using a Test Execution Report as part of the Test Report deliverable.
A typical form for reporting the pass/fail result of the execution of a test is shown below:

Table 12 Test Execution Record template

elecnor deimos	Test Execution Record	Code: Date: Page:
Project Name: Project Phase:	Module: Version:	
Date:	Originator: Signature:	
Date:	QA Manager: Signature:	
Date	Principal Investigator: Signature:	
Test Procedure:		
Result:	□ Passed	□ Failed
Test purpose: Observed results:		

5 GENERAL V&V ADMINISTRATIVE PROCEDURES

During the testing activities the problems and non-conformances will be reported as feedback to the software development process regarding the technical quality of each software product and process. The tester will correct the anomaly as part of normal work in the development activities. In case that the test will be executed during a formal test campaign as regression activities, if an anomaly is found this non conformity will be handled though SPRs.

Once solved, the problem or non-conformance will be verified that the solution is adequate and their implementation (in software or documentation) is correct.

6 Software TESTS

6.1 Software Test Designs

6.1.1 Software Test Designs

In the case of unitary tests procedures and functions will be separated into two groups:

- Procedures and functions that implement a functionality or software requirement entirely on their own, relying on other modules only for providing the necessary input parameters and for consuming the outputs of the procedures. Tests will be performed, but only as far as necessary to have confidence that they work in nominal conditions, before releasing them for integration with other units. In particular, there will be no minimum requirement as regards test coverage. Instead, the tests that are necessary will be determined on a case-by-case basis by the developers.
- Procedures and functions that implement only a part of a functionality or software requirement. Tests will be performed for these sub-programmes as part of the integration of the rest of the units that complete the implementation of the functionality or software requirement.

6.1.2 Integration Test Designs

Integration testing means taking the components that have been tested successfully at unit level (or have been deferred to the integration tests) and then testing them together to verify that they properly communicate with each other and correctly perform the job that is shared between them. This is the emphasis and added value of integration tests – they should not merely repeat the work of the lower level testing.

In the case of integration tests, the objective is to perform integration testing on the code to verify the relevant requirements. Integration tests will be performed in parallel with the activity of integration with the ENTICE environment. Several levels of integration testing might be performed, depending on the actual system. The internal results of integration testing will be kept as internal records. To do this, automatic execution of test scripts and tools will be used for automatic generation of test reports. As in the case of unit testing, integration tests shall be performed as needed, using some kind of helper tool such as JUnit or C++Unit, which allows for

easy repetition of Integration Tests. The documentation of these tests would be the test code itself.

Integration testing of software can take several forms:

- Integration of the parent-child hierarchy. This involves integrating together all those child objects that belong to the same parent object and then testing those parent object operations that are implemented by a child operation that in turn calls operations provided by its brothers and sisters. That is, test parent operations in conditions where the flow of control of the operation passes through two or more children. There is no need to test operations where the execution occurs in only one child, since that would merely repeat a test done at module level. The aim is to verify the correct interaction and cooperation between brothers and sisters that work together to implement the operations of their parent. This type of integration is itself bottom-up – the objects are integrated and verified in this way going upwards through the hierarchical architecture, ending with the verification of operations provided by the topmost, root object.
- Integration of the flow of control of each concurrent process. This method means identifying concurrent threads of execution, tracing the paths of execution through the software architectural design. For each thread, the execution is invoked starting at the root of the thread (typically the point at which the thread executes just after it is created), verifying that the execution follows the expected path and that all objects and operations along that path collaborate correctly to perform the work of the thread.

  Each thread is tested in isolation of each other concurrent thread, so for these integration tests, synchronisation calls have to be stubbed out.

6.1.3 System Test Designs

The objective of this activity is to perform system testing on the system to be delivered to verify the relevant requirements. System testing shall always be formal, and will test the system against system requirements (compliance and coverage), and possibly performance, usability, etc. Once the integration tests have verified that the architectural interfaces are correct, the functionality of the fully integrated tested application software will be validated against the corresponding system requirements. This check is always made formally for software projects at DEIMOS and sometimes with the presence of users or customer staff witnessing the test campaign, even if not required. The objective is to be sure that the resulting system is compliant with all the system requirements laid down in the specifications, and to take any Corrective Actions to ensure this status before the actual acceptance tests. System testing must be always performed in a formal way (independently of the project characteristics or customer requirements). The following steps shall be taken:

- Write down a System Test Specification following the current Validation Plan and explaining in detail the following points:

- To define the parts of the system to be tested at system level.
- To obtain RAMS analysis outputs, and apply the results for an acceptance test.
- To define the system test procedures in detail.
- To define the tools that will be used for testing.
- To specify the actual tests to be executed, including test designs, test cases, and test procedures.
- To indicate where the test results will be stored.

6.1.4 Acceptance Test Designs

The objective of this activity is to perform acceptance testing on the delivered system to validate the system against the requirements. Acceptance testing shall always be formal, and with customer witnessing, and will test the delivered and installed system against user requirements, and possibly performance, usability, etc. Once the system has been successfully tested, i.e. the system tests have verified that all requirements in the specifications are met.

Formal acceptance tests will be performed at DEIMOS' premises. The acceptance tests are usually a subset of all system tests (possibly the same set). DEIMOS will design test scenarios, i.e. hypothetical stories to help the tester work through a complex problem or test system, tracing back to the use case scenarios presented in Section 2.1. These scenarios will be comprised by a set of test cases, i.e. a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do.

Those non-functional requirements, not included in any test scenario coming from the use case analysis (mainly for functional requirements) will have a dedicated acceptance test case.

Written evidences of approval to final EOD prototype must be produced. For this, the same steps need to be done as described in the system test design section 6.1.3, but applied at acceptance level.

7 Bibliography and references

The bibliography referenced throughout the text is described in the following table.

J. Becedas, Rubén Pérez, Jose Julio Ramos, Gabor Kecskemeti, Attila Kertesz, Attila Marosi, Zsolt Nemeth, Thomas Fahringer, Radu Prodan, Simon Osttermann, Ennio Torre, Vlado Stankovski, Salman Taherizadeh, Jernej Trnkoczy, Ignacio García, Mercedes Castaño, Ignacio Campos, Craig Sheridan, Darren Whigham. (2015).Requirements Specification for ENTICE Environment. ENTICE Consortium, c/o Elecnor Deimos, http://www.entice-project.eu.
Vlado Stankovski, Matevž Breška, Filip Kralj (2015). Data Management Plan. ENTICE Consortium, c/o University of Ljubljana, http://www.entice-project.eu.

8 Abbreviations/ Glossary

GS – Ground Segment

PE – Project Engineer

PI – Principal Investigator

RAMS – Reliability, Availability, Maintainability and Safety.

SC – Scenario

SPAE – Software Product Assurance Engineer

SPR – Software Problem Report

SUT – Software Under Test

SW – Software

TM – Technical Manager

VM – Virtual Machine

VMI – Virtual Machine Image