ECSEL Research and Innovation actions (RIA)

AMASS

Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

Integrated AMASS platform (a)
D2.6

Work Package:	WP2 Reference Architecture and Integration
Dissemination level:	PU = Public
Status:	Final
Date:	28 March 2017
Responsible partner:	Morayo Adedjouma/ Bernard Botella (CEA)
Contact information:	{morayo.adedjouma, bernard.botella } AT cea.fr
Document reference:	AMASS_D2.6_WP2_CEA_V1.0

This deliverable is part of a project that has received funding from the ECSEL JU under grant agreement No 692474. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and from Spain, Czech Republic, Germany, Sweden, Italy, United Kingdom and France.

Contributors

Names	Organisation
M. Adedjouma, B. Botella	Commissariat a L'energie Atomique et aux Energies Alternatives
A. Debiasi	Fondazione Bruno Kessler
P. Böhm	AIT Austrian Institute of Technology
L. Alonso, B. López	The REUSE Company
Alejandra Ruiz	TECNALIA Research & Innovation
S. Baumgart	Ansys Medini Technologies

Reviewers

Names	Organisation
S. Medawar, D. Scholle, M. Tillman, S. Skogby (Peer reviewers)	ALTEN SE
Garazi Juez (Peer reviewer)	TECNALIA Research & Innovation
J.L. de la Vara	Universidad Carlos III de Madrid
Barbara Gallina	Maelardalen Hoegskola
Cristina Martínez (Quality Manager)	TECNALIA Research & Innovation

TABLE OF CONTENTS

List of Figures

List of Tables

Executive Summary

This deliverable (D2.6) is the first one from the Task 2.4 AMASS Platform Validation. It concerns the AMASS Open Tool Platform, which will be one of the main results of the AMASS project. This platform corresponds to a collaborative tool environment supporting CPS assurance and certification. It represents a concrete implementation of the AMASS Reference Tool Architecture, with a capability for evolution and adaptation, which will be released as an open technological solution by the AMASS project.

To reach this goal, the AMASS Consortium has decided to follow an incremental approach by developing rapid and early prototypes in three iterations. This deliverable concerns the first prototyping iteration called Prototype Core, regrouping the AMASS Platform Basic Building Blocks, that are be aligned, merged and consolidated at Technology Readiness Level (TRL) 4 (technology validated in laboratory).

The AMASS platform is composed of a set of tools providing the functionalities described in the AMASS deliverable D2.2 (AMASS Reference Architecture, first prototype). This first prototype has been built upon three pre-existing toolsets from the OpenCert project [6], the CHESS Project (Polarsys Platform) [5] and the EPF (Eclipse Process Framework) Project [7]. The components composing this first prototype are the System Component Specification, the Assurance Case Specification, the Evidence Management, the Compliance Management and the Data Manager.

The Prototype Core has been released (as source and as binaries) and two manuals have been provided with it. The Developer Guide dedicated to the AMASS Platform developers and the User Manual that targets AMASS Platform users.

The current deliverable first describes the architecture of the Prototype Core, and then presents the validation activities that have been conducted on it.

This validation has been based on an analysis of the requirements and corresponding functionalities, planned for basic building blocks constituting the Prototype Core, defined in D2.1 [10] and usage scenarios defined in D2.2 [11] in order to refine these items into test cases that are compatible with the current developments of the AMASS platform. From this analysis we defined 32 test cases to test and validate the 34 implemented functionalities of AMASS Prototype Core. These test cases have been executed by three AMASS partners.

Globally the results of tests are satisfactory, only some functionalities concerning Assurance Case Management need to be completed. From these results, the main recommendations concern the platform development process for the next iterations, asking for stable and consistent versions of the tools and the documentation before the beginning of the validation, more traceability between requirements, use cases and developed functionalities, and requiring methodological guidelines.

1. Introduction

1.1 Scope

AMASS will create and consolidate a de-facto European-wide assurance and certification open tool platform, ecosystem and self-sustainable community spanning the largest CPS vertical markets. The ultimate aim is to lower certification costs in face of rapidly changing product features and market needs. This will be achieved by establishing a novel holistic and reuse-oriented approach for:

- architecture-driven assurance fully compatible with standards such as AUTOSAR and IMA;
- multi-concern assurance for example compliance demonstration, impact analyses, and compositional assurance of security and safety aspects;
- seamless interoperability between assurance/certification and engineering activities along with third-party activities (external assessments, supplier assurance);
- cross/intra-domain re-use of, for instance, semantic standards and product/process assurance.

The AMASS tangible expected results are:

a) The **AMASS Reference Tool Architecture**, which will extend the OPENCOSS [1] and SafeCer [2] conceptual, modelling and methodological frameworks for architecture-driven and multi-concern assurance, as well as for further cross-domain and intra-domain reuse capabilities and seamless interoperability mechanisms (e.g. based on Open Services for Lifecycle Collaboration (OSLC)[1] specifications).

b) The **AMASS Open Tool Platform**, which will correspond to a collaborative tool environment supporting CPS assurance and certification. This platform represents a concrete implementation of the AMASS Reference Tool Architecture, with a capability for evolution and adaptation, which will be released as an open technological solution by the AMASS project. AMASS openness is based on both standard OSLC Application programming interfaces (APIs) with external tools (e.g. engineering tools including V&V tools) and on open-source release of the AMASS building blocks.

c) The **Open AMASS Community**, which will manage the project outcomes for maintenance, evolution and industrialization. The Open Community will be supported by a governance board, and by rules, policies, and quality models. This includes support for AMASS base tools (tool infrastructure for database and access management, among others) and extension tools (enriching AMASS functionality). As Eclipse Foundation is part of the AMASS consortium, the PolarSys/Eclipse community [4] is a strong candidate to host AMASS.

To achieve these results, the AMASS Consortium has decided to follow an incremental approach by developing rapid and early prototypes in three iterations:

1. During the **first prototyping** iteration (Prototype Core), the AMASS Platform Basic Building Blocks, will be aligned, merged and consolidated at TRL 4[2] (technology validated in laboratory).
2. During the **second prototyping** iteration (Prototype P1), the single AMASS-specific Building Blocks will be developed and benchmarked at TRL 4.

[1] https://open-services.net

[2] In the context of AMASS, the EU H2020 definition of TRL is used, see http://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2016_2017/annexes/h2020-wp1617-annex-g-trl_en.pdf

3. Finally, at the **third prototyping** iteration (Prototype P2), all AMASS building blocks will be integrated in a comprehensive toolset operating at TRL 5 (technology validated in relevant environment).

1.2 Purpose of the deliverable

This deliverable is the first one from the Task 2.4 AMASS Platform Validation. The purpose of this deliverable is to serve as a complementary to the Prototype Core. First, it provides a summarised version of the implementation work that has been done related to the basic building blocks implementation and the integration between them based on the reference architecture that was envisioned for the platform in deliverable D2.2 [11]. This document presents the different blocks and the platform architecture.

In a second part, this deliverable presents the testing and validation activities of the AMASS platform that correspond to the scope of Prototype Core, in order to check the global functionality of the platform according to the requirements defined in WP2, T2.1.

In this context, we performed an analysis of the functionalities planned for basic building blocks constituting the Prototype Core defined in D2.1 [10] and usage scenarios defined in D2.2 [11] in order to refine these items into test cases that are compatible with the current developments of the AMASS platform. Additional test cases have also been defined to check the correctness of the implementation against the AMASS User Manual [8]. The manual execution of the test cases enables us to provide direct feedback regarding implementation status and potential further enhancements for the next iteration.

The results of testing the Prototype Core and the validation team feedback will allow WP1 and T1.4 to start a fair assessment of: 1) how the objectives of the case studies are met, 2) which applications perform best, and consequently, have the biggest market potential, and 3) which aspects can be improved.

1.3 Relations to others deliverables

D2.6 is related to other AMASS deliverables:

- D2.1 [10] (Business cases and high-level requirements) defines the business models of the AMASS solutions as well as the requirements to be met by the WP3, WP4, WP5, WP6 technical AMASS work packages.
- D2.2 [11] (AMASS Reference Architecture (a)) describes the overall architecture of the AMASS platform including needs from the case studies that must be covered by the platform.
- D3.4 [12] (Prototype for Architecture-Driven Assurance (a)), D4.4 [13] (Prototype for multi-concern assurance (a)), D5.4 [14] (Prototype for seamless interoperability (a)) and D6.4 [15] (Implementation for Cross-Domain and Intra-Domain Reuse (a)) define the development of a tooling framework to support the AMASS platform first prototype. These deliverables describe the tool whose testing is reported in D2.6.
- The AMASS User Manual [8] provides a guide on how to use the AMASS platform.
- The AMASS Developer Guide [9] provides a guide on how to set up the development environment and the tools integrated in the AMASS platform.

Finally, D2.2 deliverable [11] and the AMASS User Manual [8] have been the main reference documents from which new test cases have been derived, so that the features described there can be validated.

1.4 Structure of the document

This deliverable is structured as follows: Section 2 is a presentation of the AMASS basic building blocks and of the tooling architecture and technologies used to implement them. Section 3 describes the testing and validation procedure. Section 4 contains the implementation status of the functionalities for the Prototype

Core, the definitions of the test cases that have been defined to evaluate them and the results of execution of these test cases. Section 5 provides a synthesis of the validation results of the Prototype Core and some recommendations to be considered for the next version of the platform. Appendix A provides a detailed status of the platform implementation.

2. AMASS Platform Architecture

2.1 Conceptual Architecture

A general top-level architecture of the AMASS platform has been designed as an effort done in D2.2 [11].

As part of the overall platform, the **AMASS Prototype Core** is the result of merging existing technologies from OPENCOSS [1] and SafeCer [2], and other related project such as CHESS [3]. This Prototype Core includes basic building blocks composed of tools for specification of system components, specification of assurance cases, evidence management, compliance management, user access management and data management, as well as the Common Assurance and Certification Metamodel (CACM) that merges an evolution of OPENCOSS CCL (Common Certification Language) and SafeCer metamodels.

Figure 1 provides a high-level picture of the AMASS Reference Tool Architecture (ARTA) where the basic building blocks constituting the Prototype Core are surrounded by a red dash-line.

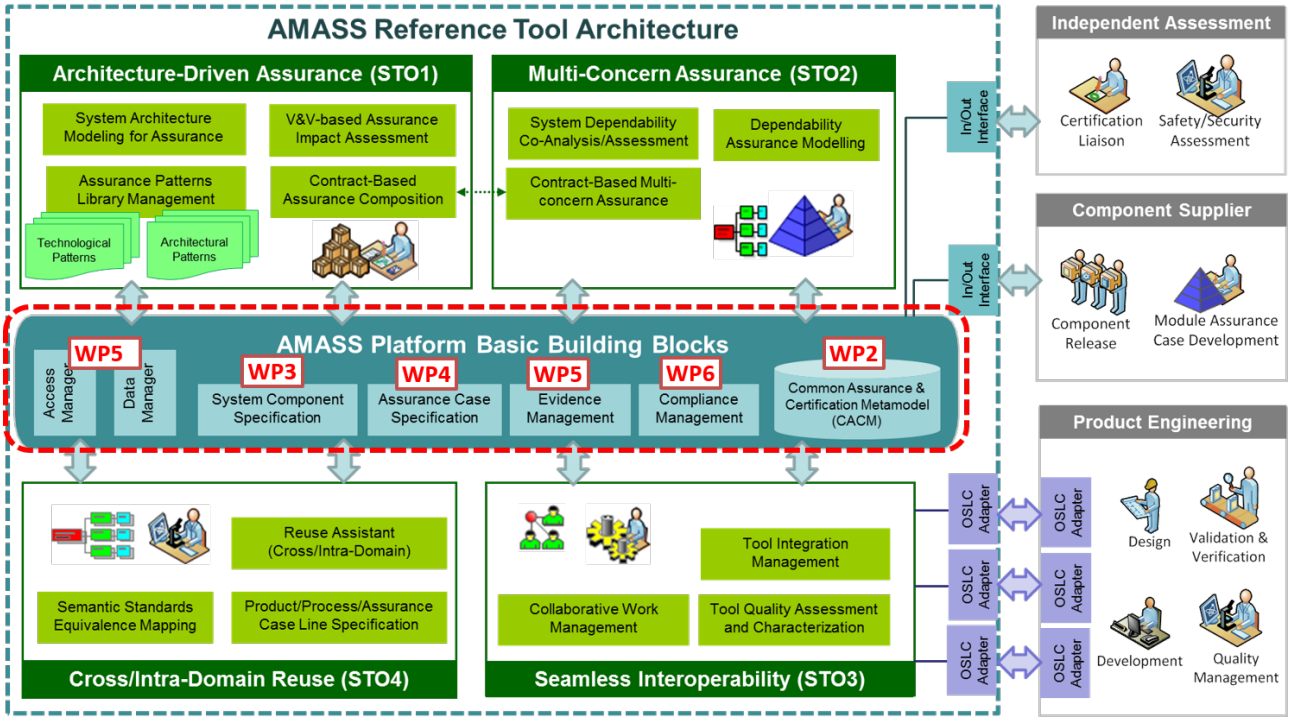

Figure 1. Scope of the AMASS Prototype Core in the overall AMASS Platform

2.2 Implementation Architecture

The designed architecture of the AMASS Prototype Core has been implemented in the scope of the T3.3, T4.3, T5.3 and T6.3 Tasks. Figure 2 presents the overall picture of the implementation architecture of the AMASS platform software building blocks and the communication between them.

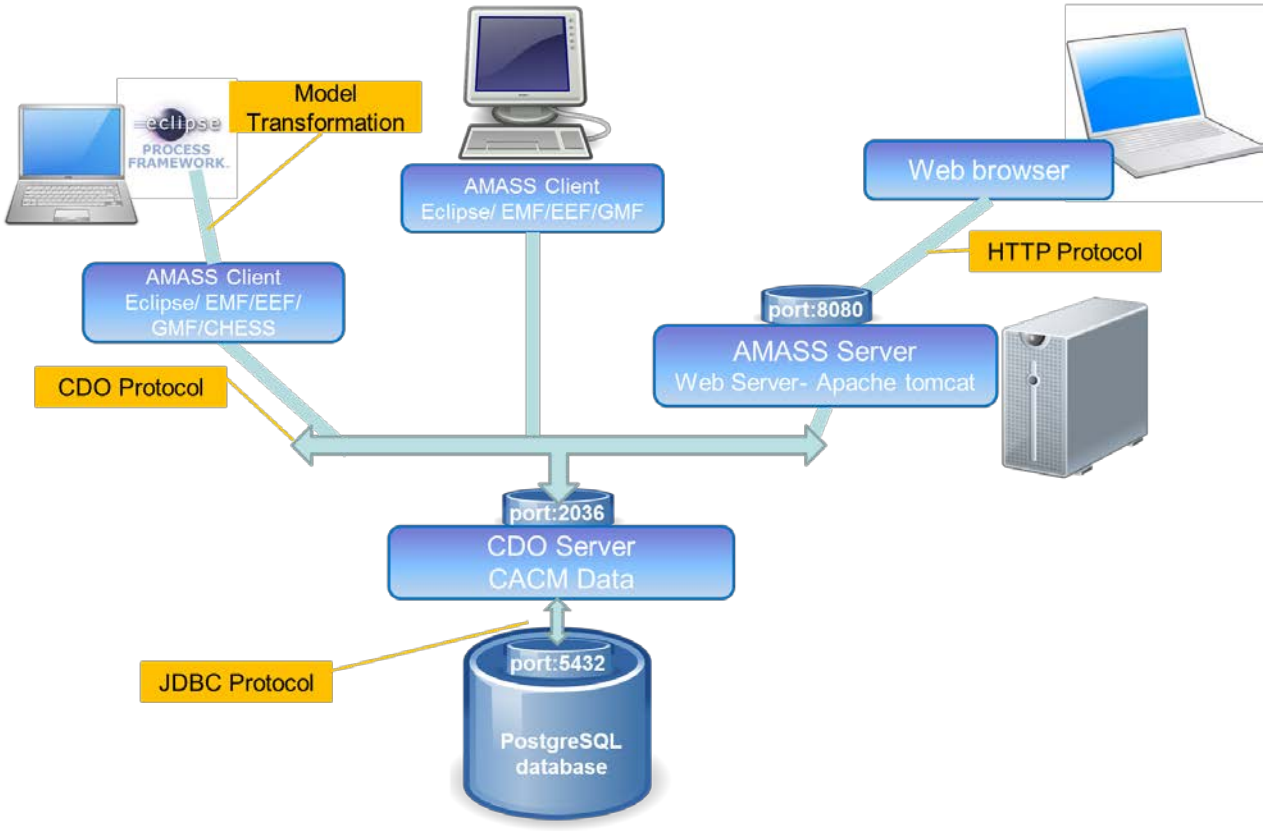

Figure 2. AMASS Implemented Architecture

The AMASS platform is composed of a set of tools providing the functionalities described in the AMASS deliverable D2.2 (AMASS Reference Architecture, first prototype). This first prototype has been built upon three pre-existing toolsets:

1. Tools from the pre-existing OpenCert project [6].
2. Tools from the CHESS Project (Polarsys Platform) [5].
3. Tools from the EPF (Eclipse Process Framework) Project [7].

The Prototype Core which integrated previous developments can be decomposed into the following main building blocks:

1. AMASS clients - facilitating data editing.
2. AMASS web server - facilitating data reporting.
3. AMASS data storage - used by both clients and the server

Figure 3 presents an overall picture of the implementation architecture of the basic building blocks and the communication between them for Prototype Core. Note that the implementation of the Access Manager basic building block has been postponed to subsequent prototype iterations.

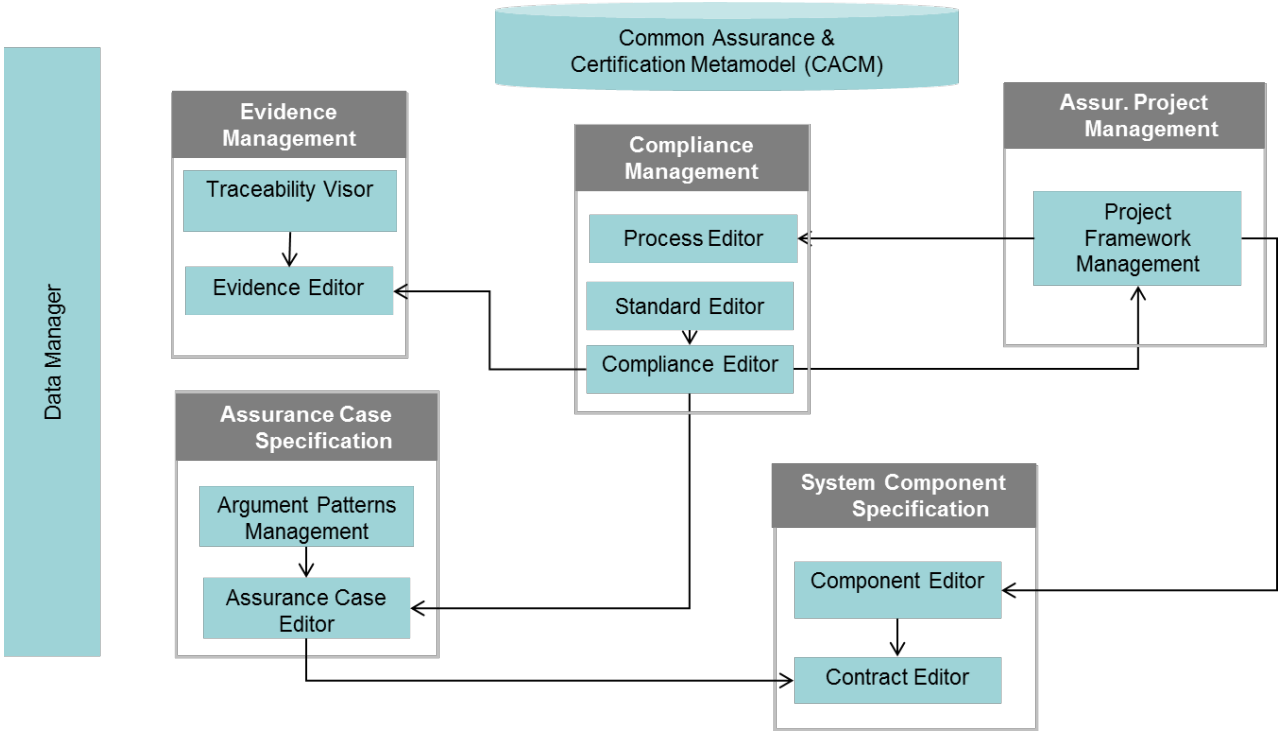

Figure 3. AMASS Reference Architecture focused on the Prototype Core basic building blocks

Looking into the AMASS client implementations, the baseline alternatives and technologies chosen for the implementation of the platform are the following:

- The **System Component Specification** basic building block reuses SafeCer and CHESS project technologies based on UML metamodel and profile mechanism through Papyrus tool editor and its CHESS extensions. Previous developments have been updated in order to support the CDO communication protocol and the Assurance Project Management block connects with this block.

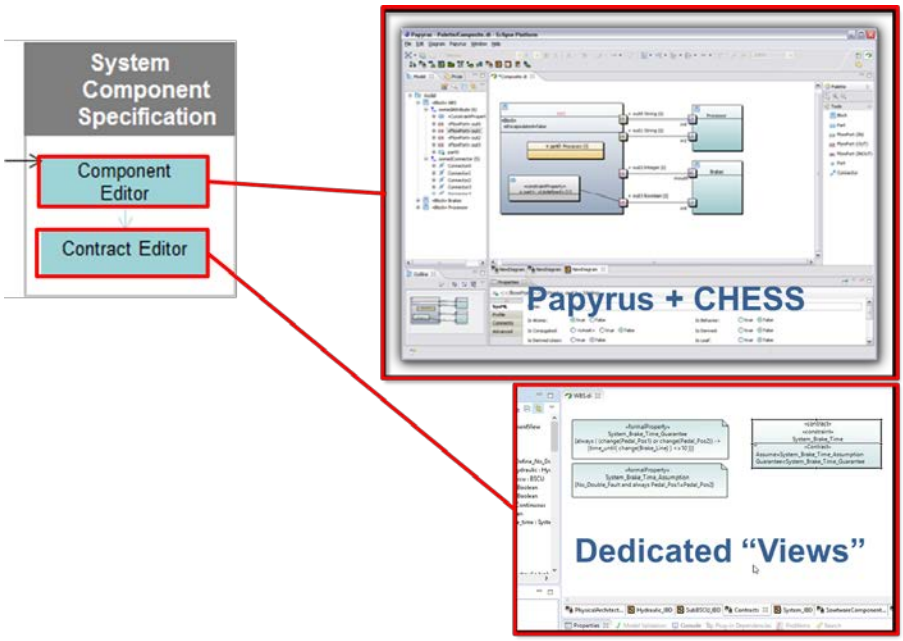

Figure 4. AMASS System Component Specification Block with screenshots associated

- The **Assurance Case Specification** basic building block reuses the argumentation tool based on the CCL (Common Certification Language) metamodel from the OPENCOSS project and provides a GSN graphical notation support like in the SafeCer project.

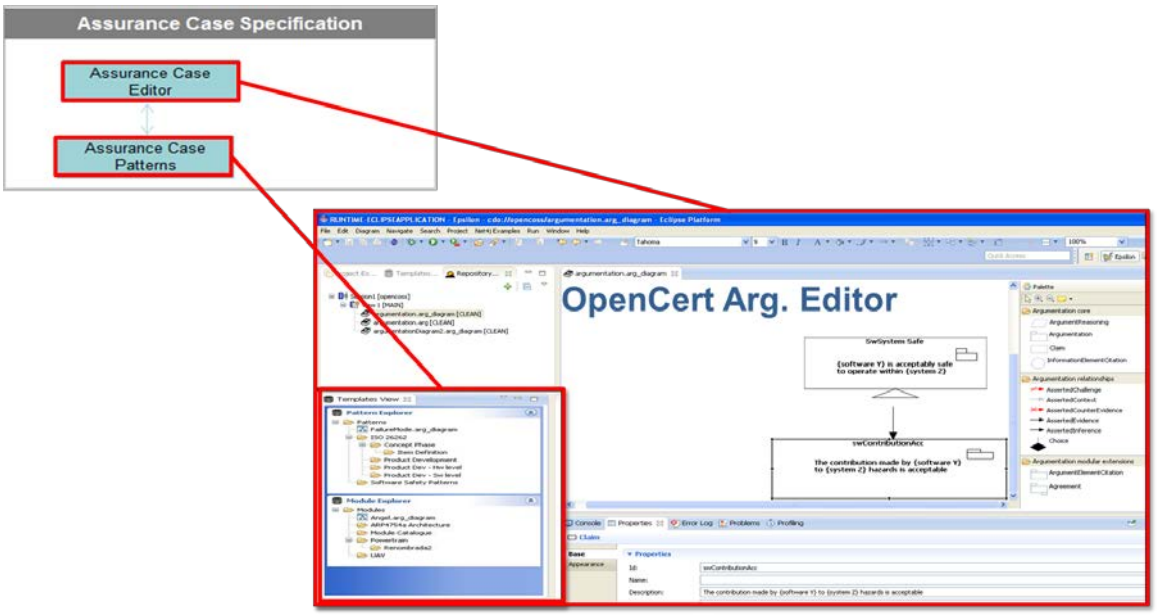

Figure 5. AMASS Assurance Case Specification Block with screenshots associated

- The **Evidence Management** basic building block reuses the evidence management tool defined in the OPENCOSS project.

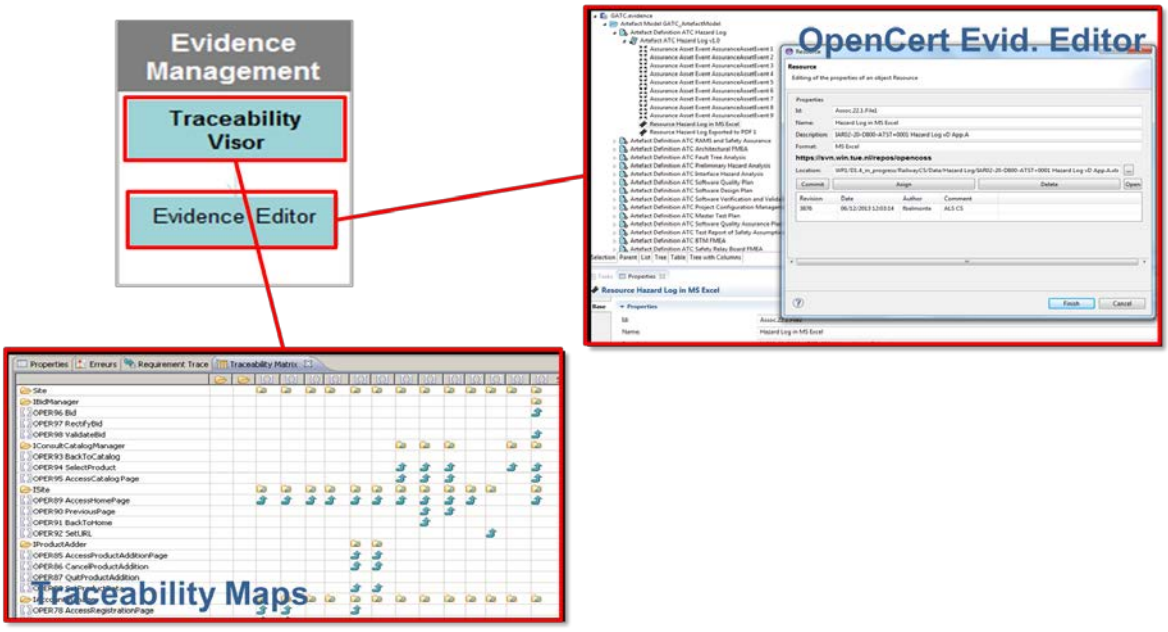

Figure 6. AMASS Evidence Management Block with screenshots associated

- The **Compliance Management** basic building block uses the Eclipse Process Framework (EPF) tool (called EPF Composer) for standards and processes modelling from the SafeCer project together with the CCL metamodel and the web-based compliance checks and reports solutions from the OPENCOSS project. We have developed specific transformations in order to get information from Process Model created in the EPF composer into the standard editor reused from OPENCOSS.

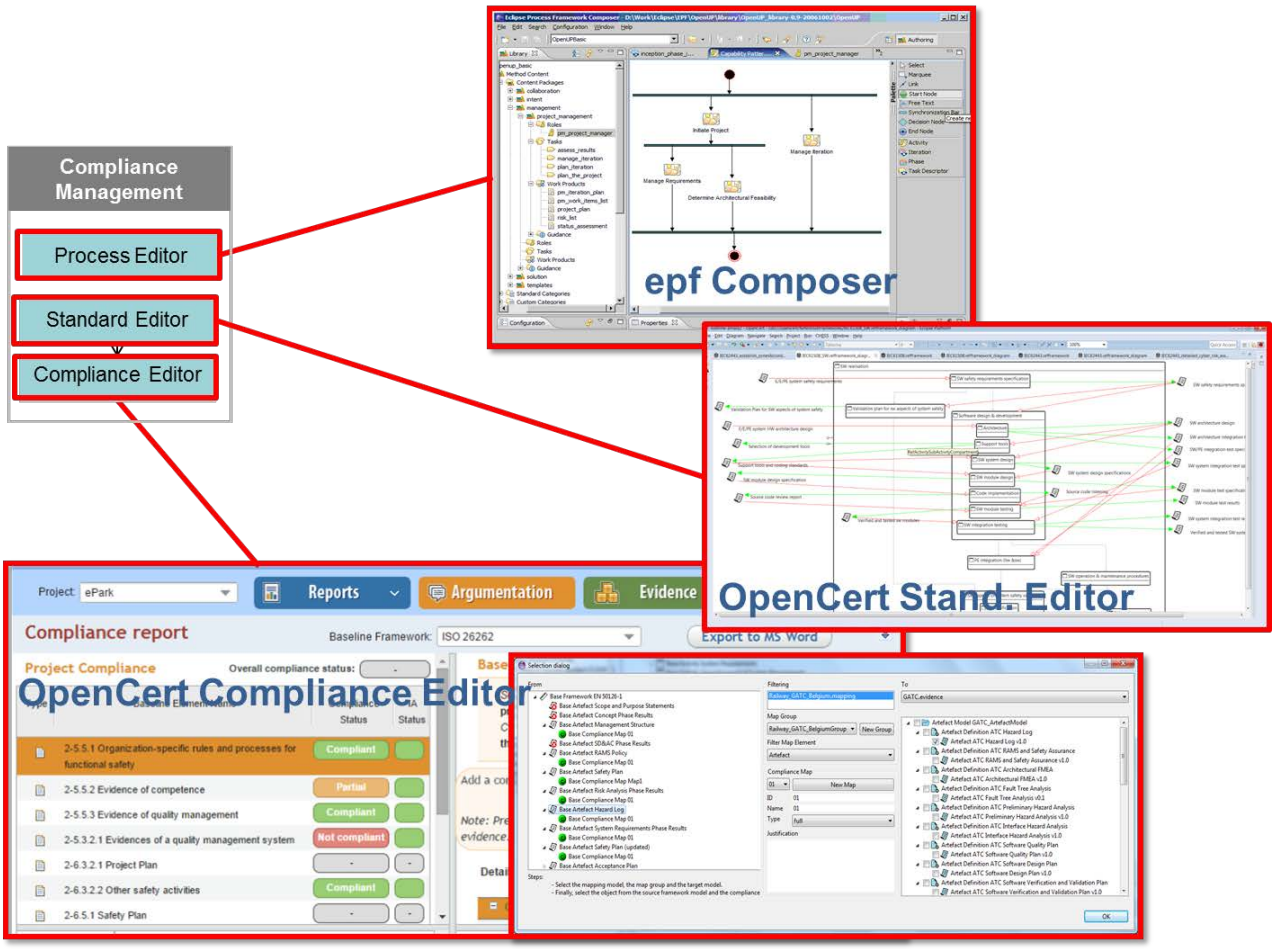

Figure 7. AMASS Compliance Management Block with screenshots associated

- The **Data Manager** basic building block supports file-based and CDO-based data storage.

2.3 AMASS Platform Prototype Core

Together with this deliverable, the Prototype Core has been released. The source code for this prototype is available at https://services.medini.eu/svn/AMASS_source [3].

Together with the source code, two binaries have been released. The first one was released at the time the testing execution was being done and the second one was released after solving some errors and bugs that were found during the validation. The binaries are available at:
https://services-medini.kpit.com/AMASS/browser/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/Tools/OpenCertCHESS

Two manuals were also developed during the Prototype Core implementation. The first manual is the Developer guide and targets the AMASS Platform developers. It was written at the same time of the implementation in a collaborative way by the developers and validated among them. The second manual is the User manual and targets the AMASS Platform users as the desirable audience. It has also been used during this period by Task 2.4 participants to specify test cases and execute them. Some comments and

[3] The AMASS SVN code repository is open to AMASS partners with the same credentials as the SVN document repository. In case that people outside the project need access, please contact the AMASS Project Manager (huascar.espinoza@tecnalia.com)

feedback from these readers have been used as input to improve the manual before been released to the people involved in the AMASS use cases. Both manuals are available at:

- Developer Guide [9] - https://services-medini.kpit.com/AMASS/browser/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/AMASS_Prototype1_DeveloperGuide.doc
- User Guide [8] - https://services-medini.kpit.com/AMASS/browser/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/AMASS_Prototype1_UserManual.docx

3. Testing and Validation Methodology

This section presents the overall methodology for validation of AMASS results. This methodology aims to validate that the AMASS Prototype Core platform satisfies its requirements and checks the system behaviour against needs from user and case studies (see D2.1 [10] and D2.2 [11] deliverables). Figure 8 presents the overall testing and validation methodology.

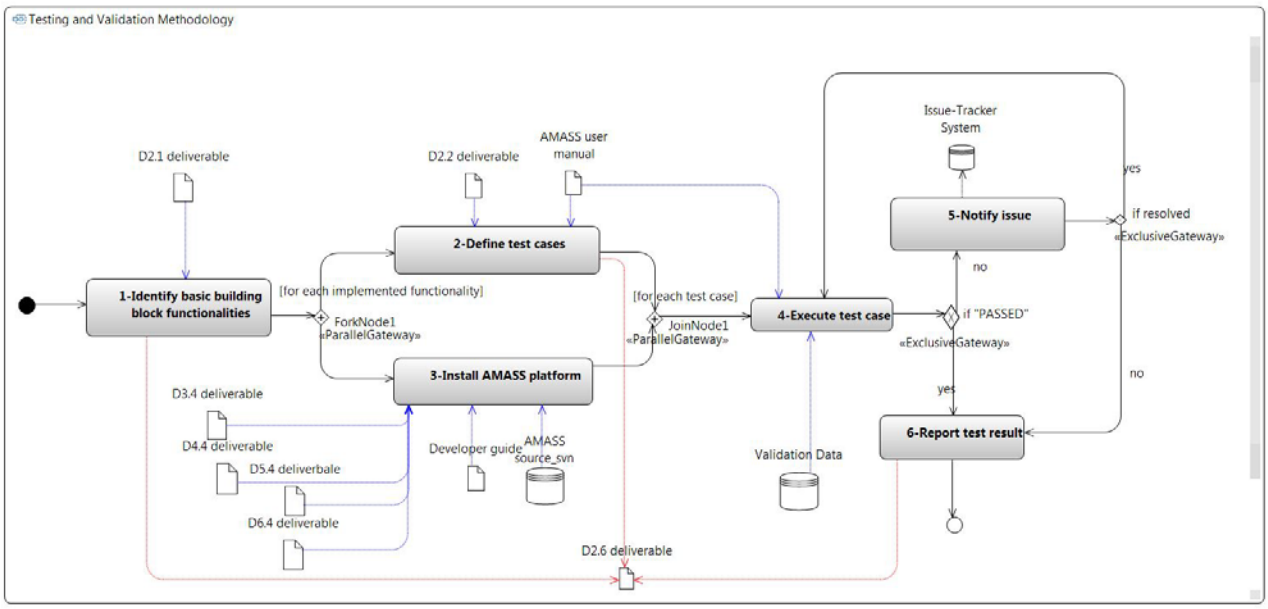

Figure 8. AMASS testing and validation methodology

In step 1, the Prototype Core functionalities have been collected from D2.1 deliverable. In step 2, we define the test cases corresponding to the implemented functionalities. The test cases are mainly based on the scenarios defined in the use cases of D2.2 deliverable. The test cases aim to provide concrete details about how AMASS will be used and when such usage can be regarded as successful. The test cases have been also traced back to the D2.1 requirements of the AMASS Prototype Core to ensure their theoretical coverage.

The D2.1 and D2.2 reference documents have been used in the versions available at the time of the test execution, they have not been always in line with the implementation status of the AMASS platform at that time. This is why we have also used the AMASS User Manual as a reference document to derive some test cases.

The specification of a test case consists of the following information:

- Test Case ID, which uniquely identifies the test case
- Scope, which provides the context and summarizes the purpose of the test case
- Functionality ID, which refers to the AMASS related requirements that must be validated
- Related use cases, which refer to the use case scenarios that are concerned
- Input, which specifies the necessary input data needed prior to execute the test case
- Steps are the execution steps to follow to run the test case
- Expected results specify the behaviour or computation results expected from the execution of the test case
- MoSCoW Priority[4] as defined for the AMASS requirements in D2.1 deliverable

[4] Must have, Should have, Could have, and Won't have but would like

In step 3, the Validation Team has installed the required material for running the test cases based on the Developer Guide, and the D3.4 [12], D4.4 [13], D5.4 [14] and D6.4 [15] deliverables. The software is installed from a SVN repository. In step 4, the validation team has executed manually the test cases, as this was the most efficient way at that development stage. We report the status of the execution of the test cases as:

- Passed: functionality that works as required
- Passed but: functionality that works but could be enhanced
- Failed: functionality that does not work

In step 5, for each test case with “Passed but” or “Failed” result status, a rationale is given to describe the problem identified in the software or the User Manual. We generated a ticket within the selected Issue-Tracker system for such test cases to report the problem to the Implementation Team. When the test cases have been resolved, they are executed again to update their status.

In step 6, we report the final status of the testing and validation in D2.6 deliverable.

4. Testing and Validation for System Component Specification Basic Building Block

4.1 System Component Specification Functionalities for Prototype Core

The functionalities concerning the System Component Specification basic building block are defined in D2.1 deliverable [10]. Table 1 is an excerpt of the relevant functionalities planned for Prototype Core, their implementation status and the implementation responsible. Two functionalities among 16 defined have not been implemented and are postponed for the next version of the AMASS platform prototypes.

Table 1. System Component Specification basic building block functionalities

ID	Functionality	Status	Responsible
WP3_SC_001	Browse along the different abstractions levels (system, subsystem, component)	Implemented	INT
WP3_SC_002	Move and edit along the different abstractions levels (system, subsystem, component)	Implemented	INT
WP3_SC_004	Formalize requirements with formal properties	Implemented	INT
WP3_SC_005	Provide the capability for allocating requirements to parts of the component model.	Implemented	INT
WP3_SC_006	Specify the component behavioural model	Implemented	INT
WP3_SC_007	Fault injection (include faulty behaviour of a component)	Postponed	
WP6_RA_003	Provide the capability for reuse of pre-developed components and their accompanying artefacts	Implemented	INT
WP3_CAC_002a	Associate a contract to a component	Implemented	INT
WP3_CAC_002b	Drop contract from component	Implemented	INT
WP3_CAC_003	Structure properties into contracts (assumptions/guarantees)	Implemented	INT
WP3_CAC_004	Specify the refinement of the contract along the hierarchical components architecture	Implemented	INT
WP3_CAC_012	Browse Contract status	Implemented	INT
WP3_CAC_013	Specify contracts defining the assumption and the guarantee elements	Implemented	INT
WP3_SAM_001	Trace all the assurance information with the specific component	Implemented	INT
WP3_VVA_001	Trace immediate evidence (obtained during the execution of the left-hand side of the V-model) with direct evidence (obtained during the execution of the right-hand side of the V-model).	Implemented	INT
WP3_VVA_004	Trace requirements validation checks	Postponed	

4.2 System Component Specification Test Cases

In this section, we present the set of test cases defined to validate the correct implementation of the System Component Specification basic building block of Prototype Core. The test cases have been defined based on the use case scenarios defined in the deliverable D2.2 [11] for the concerned functionalities when existing.

Table 2. Defined Test Case WP3_TC_01 for WP3_SC_001 functionality

ID	WP3_TC_01
Scope	Browse along the different abstractions levels (system, subsystem, component)
Functionality ID	WP3_SC_001
Related use cases	Use case “Specify system architecture”
Input	None
Steps	1. Use the Model Explorer View to browse along the different abstractions levels (e.g. RequirementView, SystemView, ComponentView, DeploymentView, AnalysisView) 2. Select one diagram (e.g. PhysicalArchitecture_BDD in the ModelSystemView) from the Model Explorer View to get access to the diagram information
Expected results	A set of graphical views (e.g. Model-based Editor View, Properties View, Outline View) provide information about a specified architecture. They are updated with respect to the selected diagram
Priority	Must

Table 3. Defined Test Case WP3_TC_02 for WP3_SC_002 functionality

ID	WP3_TC_02
Scope	Move and edit along the different abstractions levels
Functionality ID	WP3_SC_002
Related use cases	Use case “Specify system architecture”
Input	None
Steps	1. Browse the model using the Model Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Create a new model (e.g. a Block Definition Diagram) 3. Use the Model-based Editor or the Model Explorer View to create model entities (e.g. packages, contracts, interfaces, etc...) 4. Use the Model-based Editor or the Model Explorer View to delete model entities (e.g. packages, contracts, interfaces, etc...) 5. Use the Model Explorer View to delete the model
Expected results	A new diagram is created
Priority	Must

Table 4. Defined Test Case WP3_TC_19 for WP3_SC_006 functionality

ID	WP3_TC_19
Scope	Specify the component behavioural model
Functionality ID	WP3_SC_006
Related use cases	Use case “Specify system architecture”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Create a StateMachine Diagram from the Model Explorer View 3. Use the Model-based Editor View and the related Palette to create, associate to the component, and model the state machine that defines the component behaviour
Expected results	A new state machine diagram is defined
Priority	Should

Table 5. Defined Test Case WP3_TC_25 for WP6_RA_003 functionality

ID	WP3_TC_25

Scope	Provide the capability for reuse of pre-developed components and their accompanying artefacts
Functionality ID	WP6_RA_003
Related use cases	Use case “Specify system architecture”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Create a Block Definition Diagram (e.g. newPhisicalArchitecture_BDD) from the Model Explorer View 3. Create a System component (e.g. System2) using the Palette 4. In the Model Explorer View select the System component and create an Internal Block Diagram 5. Create a Part component inside the System component using the Palette 6. In the Properties View - UML tab, change the type selecting the element to reuse (e.g. the subsystem BSCU)
Expected results	A Diagram with a reused component
Priority	Must

Table 6. Defined Test Case WP3_TC_14 for WP3_SC_004 functionality

ID	WP3_TC_14
Scope	Formalize requirements with formal properties
Functionality ID	WP3_SC_004
Related use cases	Use case “Specify system architecture”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the SoftwareContract package) 2. Open the diagram (e.g. SoftwareContracts_CD) from the Model Explorer View 3. Create a “Formal Property” entity using the Palette. 4. Select the created entity and formalize the requirement using the property view
Expected results	The formalized requirement in a textual area of the Property View and in the Model Explorer View
Priority	Must

Table 7. Defined Test Case WP3_TC_16 for WP3_SC_005 functionality

ID	WP3_TC_16
Scope	Provide the capability for allocating requirements to parts of the component model
Functionality ID	WP3_SC_005
Related use cases	Use case “Specify system architecture”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Open the diagram (e.g. PhysicalArchitecture_BDD) from the Model Explorer View 3. In the Palette of the Model-based Editor View drag an entity with type Constraint (e.g. CriticalityLevel) over a component (e.g. BSCU). 4. Select the already created constraint 5. In the Property View – UML Tab, text area Specification, specify the

	requirement as formal condition
Expected results	The formalized requirement assigned to a specific component
Priority	Must

Table 8. Defined Test Case WP3_TC_30 for WP3_CAC_002a functionality

ID	WP3_TC_30
Scope	Associate a contract to a component
Functionality ID	WP3_CAC_002a
Related use cases	Use case “Assign a contract to the component”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Select the diagram (e.g. PhysicalArchitecture_CD) from the Model Explorer View 3. Select Contract from the Palette and click on the diagram. 4. Give a proper name to the Contract 5. Create a ContractProperty inside the Block/Component (e.g. BSCU) 6. In the Property View – UML Tab, type the just created ContractProperty with the Contract
Expected results	A component updated with a property that represents the contract assignment
Priority	Must

Table 9. Defined Test Case WP3_TC_31 for WP3_CAC_002b functionality

ID	WP3_TC_31
Scope	Drop contract from component
Functionality ID	WP3_CAC_002b
Related use cases	Use case “Assign a contract to the component”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Open the diagram (e.g. PhysicalArchitecture_CD) from the Model Explorer View 3. Select a component (e.g. BSCU) 4. Delete the ContractProperty associated to the contract
Expected results	The updated diagram without the assignment of contract
Priority	Must

Table 10. Defined Test Case WP3_TC_29 for WP3_CAC_004 functionality

ID	WP3_TC_29
Scope	Specify the refinement of the contract along the hierarchical components architecture
Functionality ID	WP3_CAC_004
Related use cases	Use case “Refine component contract”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Select the diagram (e.g. PhysicalArchitecture_CD) from the Model Explorer View 3. Select a Block (e.g. BSCU) 4. Select a ContractProperty and use the Properties view – Profile Tab to

	perform a contract refinement 5. The information about the refinement is set in the RefinedBy attribute of the ContractProperty stereotype
Expected results	The block is updated with the information about the contract refinement
Priority	Must

Table 11. Defined Test Case WP3_TC_05 for WP3_CAC_013 functionality

ID	WP3_TC_05
Scope	Specify contracts defining the assumption and the guarantee elements
Functionality ID	WP3_CAC_013
Related use cases	Use case “Structure properties into contracts”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Select the diagram (e.g. PhysicalArchitecture_CD) from the Model Explorer View 3. Select the contract (e.g. BSCU_Safety) 4. Use the Properties view – Profile Tab to bind the existing Formal Properties as contract’s assumption and guarantee
Expected results	The updated diagram with the contract composed to linked assumption and guarantee property
Priority	Should

Table 12. Defined Test Case WP3_TC_15 for WP3_CAC_003 functionality

ID	WP3_TC_15
Scope	Structure properties into contracts (assumptions/guarantees)
Functionality ID	WP3_CAC_003
Related use cases	Use case “Structure properties into contracts”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Select the diagram (e.g. PhysicalArchitecture_CD) from the Model Explorer View 3. Select a Contract (e.g. BSCU_Safety) 4. Use the Properties view – Contracts Tab to edit the assumption and guarantee property of the contract
Expected results	The updated diagram with the contract composed to edited assumption and guarantee property
Priority	Must

Table 13. Defined Test Case WP3_TC_03 for WP3_SAM_001 functionality

ID	WP3_TC_03
Scope	Trace all the assurance information with the specific component
Functionality ID	WP3_SAM_001
Related use cases	Use case “Trace contract to evidence and assurance case”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Select the Contract or FormalProperty in the model to link (e.g. System_Brake_Time in the Contracts diagram)

	3. Open the OpenCert tab in the Properties view. The OpenCert tab shows the relationships stated above between Contract/FormalProperty, Claim and Artefact 4. Pin the Properties view 5. Drag the Claim/Artefact from the Project Explorer view to claim/artefact property area of the OpenCert tab
Expected results	The creation of a link. To delete the link, select the entity in the OpenCert tab and click on the delete button available on the right of the OpenCert tab.
Priority	Must

Table 14. Defined Test Case WP3_TC_33 for WP3_CAC_012 functionality

ID	WP3_TC_33
Scope	Browse Contract status
Functionality ID	WP3_CAC_012
Related use cases	Use case “Browse component contracts status”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Select the Contract in the model (e.g. BSCU_CMD_Time in the Contracts diagram) 3. Open the Properties view – OpenCert Tab. It shows the relationships stated above between Contract/FormalProperty, Claim and Artefact. 4. Check the contract status
Expected results	A list of artefacts that support the contract
Priority	Must

Table 15. Defined Test Case WP3_TC_06 for WP3_VVA_001

ID	WP3_TC_06
Scope	Trace immediate evidence with direct evidence
Functionality ID	WP3_VVA_001
Related use cases	Use case “Refine component contract”
Input	None
Steps	1. Browse the model using the Project Explorer View (e.g. go inside the PhysicalArchitecture package) 2. Select the Contract or FormalProperty in the model to link (e.g. System_Brake_Time in the Contracts diagram) 3. Open the Properties view – OpenCert Tab. The OpenCert tab shows the relationships stated above between Contract/FormalProperty, Claim and Artefact 4. Pin the Properties view 5. Drag the Claim/Artefact from the Project Explorer view to claim/artefact property area of the OpenCert tab
Expected results	A link is created. To delete the link, select the entity in the OpenCert tab and click on the delete button available on the right of the OpenCert tab.
Priority	Should

4.3 System Component Specification Test Results

Table 16 presents, for each test case defined for the implemented System Component Specification basic building block functionalities, the results of the execution, the status and the validation responsible.

The instructions for installing the used testing environment are described in the AMASS Developer Guide [9]. The System Component Specification functionalities provided by Prototype Core are detailed in the AMASS User Manual [8]. The validation data used to perform the execution of the test cases have been restored from a database backup[5] and an existing Eclipse project[6] provided by the implementation responsible.

The test cases have been performed with the following machine configuration: Operating system: Windows 10 Enterprise (64 bits), Processor: Intel Core i7-56000U, CPU @ 2.60 GHz, RAM: 16 GB. All the test cases have been passed for the building block.

Table 16. Test results for the implemented System Component Specification functionalities

Test Case ID	Execution Results	Status	Rationale	Responsible
WP3_TC_01	A set of graphical views provide information about a specified architecture. They are updated with respect to the selected diagram.	Passed		FBK
WP3_TC_02	A new diagram with new entities are created	Passed		FBK
WP3_TC_19	A new state machine diagram is defined	Passed		FBK
WP3_TC_25	A Diagram with a reused component	Passed		FBK
WP3_TC_14	The formalized requirement in a textual area of the Property View and in the Model Explorer View	Passed		FBK
WP3_TC_16	The formalized requirement assigned to a specific component	Passed		FBK
WP3_TC_30	A component updated with a property that represents the contract assignment.	Passed		FBK
WP3_TC_31	The updated diagram without the assignment of contract.	Passed		FBK
WP3_TC_32	The updated diagram with the reassigned contract	Passed		FBK
WP3_TC_29	The block is updated with the information about the contract refinement	Passed		FBK
WP3_TC_05	The updated diagram with the contract composed to linked assumption and guarantee property	Passed		FBK
WP3_TC_15	Structure properties into contracts	Passed		FBK
WP3_TC_03	The creation of a link. To delete the link, select the entity in the OpenCert tab and click on the delete button available on the right of the OpenCert tab	Passed		FBK
WP3_TC_33	A list of artefacts that support the contract	Passed		FBK
WP3_TC_06	The creation of a link. To delete the link, select the entity in the OpenCert tab and click on the delete button available on the right of the OpenCert tab	Passed		FBK

5 The database backup used is SystemComponentSpecTest, located in the SVN repository https://services-medini.kpit.com/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/Vaditation_Data/CHESS

6 The Eclipse Project WBS_CHESS_OpenCert is located in the SVN repository https://services-medini.kpit.com/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/Vaditation_Data/CHESS

5. Testing and Validation for Assurance Case Specification Basic Building Block

5.1 Assurance Case Specification Functionalities for Prototype Core

The functionalities concerning the Assurance Case Specification basic building block are defined in the deliverable D2.1 [10]. Table 17 is an excerpt of these functionalities planned for Prototype Core, their implementation status and the implementation responsible. The IDs are, as much as possible, taken from D2.1, but some functionalities planned for Prototype Core have no direct correlation link with any D2.1 requirements and keep here their implementation IDs (WP4_4.8, WP4_4.15 and WP4_4.10). Among the eight planned functionalities for Prototype Core, two functionalities have not been implemented and are postponed for the next version of the AMASS platform.

Table 17. Assurance case Specification basic building block functionalities

ID	Functionality	Status	Responsible
WP4_ACS_001	Edit an assurance case in a scalable way	Implemented	TEC
WP4_ACS_003	Instantiate in the actual assurance case an argument pattern (concerning safety and security) selected from the list of patterns stored	Implemented	TEC
WP4_ACS_004	Semi-automatic generation of process arguments	Postponed	
WP4_ACS_005	Provide support for language formalization inside argument claims	Implemented	TEC
WP4_ACS_010	Provide the capability of generating a compositional assurance case argument	Implemented	TEC
WP4_4.8	Navigate from an evidence supporting a claim to the information about the evidence such as the evidence characterization and the actual artefact	Implemented	TEC
WP4_4.15	Let different users edit an assurance case in a collaborative manner	Postponed	TEC
WP4_4.19	Edit and store argumentation patterns for later use	Implemented	TEC

5.2 Assurance Case Specification Test Cases

This section presents the set of test cases defined to validate the correct implementation of the Assurance Case Specification basic building block of Prototype Core. Test cases have been defined based on the use case scenarios defined in the D2.2 deliverable [11] for the concerned functionalities when existing.

Table 18. Defined Test Case WP4_TC_01 for WP4_4.2 functionality

ID	WP4_TC_01
Scope	Edit an assurance case in a scalable way
Functionality ID	WP4_ACS_001
Related use cases	Use Case “Define and navigate an assurance case structure”
Input	A reference framework

Steps	1. Create an assurance project 2. Create a baseline from a big reference framework (ISO 26262) 3. Choose to create automatically the argumentation diagram 4. Browse the argument diagram elements 5. Create new elements, links 6. Update the elements 7. Delete some elements 8. Save the argumentation diagram 9. Create a diagram view 10. Drag and drop element from outline menu to diagram editor 11. Hide an element on the diagram 12. Delete an element on the diagram 13. Create a new diagram from the argumentation model
Expected results	Modified assurance case
Priority	Must

Table 19. Defined Test Case WP4_TC_03 for WP4_4.5 functionality

ID	WP4_TC_03
Scope	Provide the capability of generating a compositional assurance case argument
Functionality ID	WP4_ACS_010
Related use cases	Use Case “Define and navigate an assurance case structure”
Input	None
Steps	For every argument module: 1. Specify manually the claims set 2. Provide stated and valid assumptions applied to the claims 3. Specify contextual information to define or constraint the scope over which the arguments are assumed to be valid 4. Map claims (away goals) to the external claims (public goals) that support to (in other argument modules)
Expected results	A compositionally defined assurance case
Priority	Must

Table 20. Defined Test Case WP4_TC_04 for WP4_4.8 functionality

ID	WP4_TC_04
Scope	Connection from the supporting evidences to evidence information
Functionality ID	WP4_4.8
Related use cases	Use case “Develop Claims and Links to Evidence”
Input	Argumentation model, pieces of evidence
Steps	1. Start from an assurance case (result of WP4_TC_01) 2. Create an evidence model 3. Add artefacts in the evidence model 4. Go to the argumentation diagram 5. Edit solutions, contexts and justifications elements and add corresponding artefacts
Expected results	An argumentation model linked to evidences
Priority	Must

Table 21. Defined Test Case WP4_TC_05 for WP4_4.9 functionality

ID	WP4_TC_05
Scope	Drag and drop argumentation patterns

Functionality ID	WP4_ACS_003
Related use cases	Use case “Apply an argument pattern”
Input	An argumentation model under edition
Steps	1. Start from an existing pattern library (see WP4_TC_08) 2. Open an argumentation diagram 3. Open the template view 4. Drag and drop a pattern into this diagram 5. Arrange and edit the included elements 6. Save the diagram
Expected results	Changes in an argumentation model are registered
Priority	Must

Table 22. Defined Test Case WP4_TC_06 for WP4_4.13 functionality

ID	WP4_TC_06
Scope	Provide support for language formalization inside argument claims
Functionality ID	WP4_ACS_005
Related use cases	None
Input	An argumentation model
Steps	1. Create a vocabulary diagram 2. Add categories and terms 3. Open an argumentation diagram 4. Edit the description of the elements using the defined terms 5. Save the vocabulary on an xml file
Expected results	A vocabulary and an argumentation using it inside claims
Priority	Must

Table 23. Defined Test Case WP4_TC_08 for WP4 _4.19 functionality

ID	WP4_TC_08
Scope	Edit and store argumentation patterns for later use.
Functionality ID	WP4 _4.19
Related use cases	Use case “Define and navigate an assurance case structure”
Input	None
Steps	1. Create a general local project 2. Define a PATTERNS folder in the project 3. Define the argumentation Opencert/Argumentation preference concerning patterns directory to this folder 4. Create a new argumentation diagram (to file) in this folder 5. Edit this pattern diagram 6. Save the diagram
Expected results	Feasibility of reusing previously created argument packages
Priority	Should

5.3 Assurance Case Specification Test Results

Table 24 presents, for each test case defined for the implemented Assurance Case Specification basic building block functionalities, the results of the execution, the status, a rationale when the execution failed and finally the AMASS project partner who is responsible for the validation of the test case.

The documentation and data used to perform the test cases execution are as followed: the instructions to install the used testing environment are described in the AMASS Developer Guide [9]. The Assurance Case

Specification functionalities provided by Prototype Core are detailed in the AMASS User manual [8]. The validation data have been restored from a CDO repository[7] provided by the implementation responsible.

The test cases have been performed with the following machine configuration: Operating system: Windows 7 Enterprise (64 bits), Processor: Intel Core i7-56000U, CPU @ 2.60 GHz, RAM: 16 GB. Three test cases for the building block have successfully passed, and three test cases have failed.

Table 24. Test results for the implemented Assurance Case Specification functionalities

Test Case ID	Execution Results	Status	Rationale	Responsibility
WP4_TC_01	An argumentation model	Failed	-There are inconsistencies in terminology between GSN and CACM in the tool (for example you create a "goal", but in the properties tab it is called a "claim ", and in the cited by property too) -The COPY/PASTE functionality is not available (the lines in the edit menu are greyed) -Step 11: the contextual menu Delete from diagram is greyed - Step 13: there is no contextual menu proposed for creating the diagram	CEA
WP4_TC_03	An assurance case	Failed	Reading the user manual, we are not able to understand how the modules may be defined in different diagrams and then linked in a coherent way. We have just been able to insert GSN modular extensions concepts in a diagram	CEA
WP4_TC_04	An argumentation model linked to evidences	Passed		CEA
WP4_TC_05	An argumentation model built instantiating a predefined pattern	Passed		CEA
WP4_TC_06	A vocabulary and an argumentation using it inside claims	Failed	-We create a vocabulary, both on a file or in the remote repository. However, we do not success in associating the vocabulary to the assurance case. When we use CTRL-SPACE during claim editing, nothing happens. -The user manual explains how to load a vocabulary in xml format but does not explain how to save the	CEA

[7] The CDO repository used is **amass.tecnalia.com** (server: amass.tecnalia.com, port: 2036)

			vocabulary in an xml file	
WP4_TC_08	A pattern	Passed	There are inconsistencies in terminology between GSN and CACM in the tool in the pattern model	CEA

6. Testing and Validation for Evidence Management Basic Building Block

6.1 Evidence Management Functionalities for Prototype Core

The functionalities concerning Evidence Management basic building block are defined in the D2.1 deliverable [10]. Table 25 is an excerpt of these functionalities, keeping those planned for Prototype Core, their implementation status and the AMASS project partner who is responsible for the validation of the test case. Among the fourteen planned functionalities for Prototype Core, five functionalities have not been implemented and postponed for the next version of the AMASS platform.

Table 25. Evidence Management basic building block functionalities

ID	Functionality	Status	Responsible
WP5_EM_001	Evidence characteristics specification	Implemented	TEC
WP5_EM_002	Evidence traceability	Implemented	TEC
WP5_EM_003	Evidence change impact analysis	Implemented	TEC
WP5_EM_004	Evidence evaluation	Implemented	TEC
WP5_EM_005	Evidence information import	Implemented	TEC
WP5_EM_010	Evidence lifecycle information storage	Implemented	TEC
WP5_EM_011	Interactive evidence change impact analysis	Implemented	TEC
WP5_EM_013	Link of evidence to other assets	Implemented	TEC
WP5_EM_014	Evidence resource specification	Implemented	TEC
WP5_EM_006	Evidence information export	Postponed	
WP5_EM_008	Visualization of chains of evidence	Postponed	
WP5_EM_009	Suggestion of evidence traces	Postponed	
WP5_EM_012	Evidence trace verification	Postponed	
WP5_EM_015	Resource part selection	Postponed	

6.2 Evidence Management Test Cases

The tables in this section define the test cases to validate the correct implementation of the Evidence Management basic building block of the Prototype Core. Test cases have been defined based on the use case scenarios defined in the D2.2 deliverable [11] for the concerned functionalities when existing.

Table 26. Defined Test Case WP5_TC_01 for WP5_5.1, WP5_5.5, WP5_5.14, WP5_5.21 functionalities

ID	WP5_TC_01
Scope	Specification of the main characteristics of an artefact model and its artefacts
Functionality ID	WP5_EM_001, WP5_ EM_005, WP5_ EM_014
Related use cases	Use case “Characterise Managed Artefact, Import of artefact information from an external tool”
Input	Assurance Project
Steps	1. Open the Assurance Project 2. Configure the artefact repository for SVN 3. Create an Artefact Model for the Assurance Project 4. Add an Artefact Definition 5. Add an Artefact to the Artefact Definition

	6. Fill all the fields of the Artefact 7. Add a Resource to the Artefact 8. Specify the information of the Resource 9. Add a Value to the Artefact 10. Specify the information of the Value 11. Add a Sub-artefact to the Artefact 12. Fill the fields of the Sub-Artefact 13. Add another Artefact 14. Indicate that the precedent version of the second Artefact is the first Artefact 15. Save the Artefact Model 16. Close the Artefact Model 17. Open the Artefact Models
Expected results	It has been possible to execute all the steps, and all the information specified for the Artefact Model is available
Priority	Must

Table 27. Defined Test Case WP5_TC_02 for WP5_5.4, WP5_5.10, WP5_5.21 functionalities

ID	WP5_TC_02
Scope	Specification of lifecycle information for artefacts
Functionality ID	WP5_ EM_004, WP5_ EM_010
Related use cases	Use case "Specify Managed Artefact Lifecycle, Specification of evaluation information for artefacts"
Input	• Assurance project • Artefact Model with Artefacts, of the Assurance Project
Steps	1. Open the Artefact Model 2. Select an Artefact 3. Add an Assurance Asset Event to the Artefact 4. Specify the information of the Event 5. Select another Artefact 6. Add an Assurance Asset Evaluation to the Artefact 7. Specify the information of the Evaluation 8. Save the Artefact Model 9. Close the Artefact Model 10. Open the Artefact Models
Expected results	It has been possible to execute all the steps, and all the information specified for the Artefact Model is available
Priority	Must

Table 28. Defined Test Case WP5_TC_03 for WP5_5.2, WP5_5.3, WP5_5.11, WP5_5.21 functionalities

ID	WP5_TC_03
Scope	Specification of traceability information for artefacts and impact analysis as a result of artefact change
Functionality ID	WP5_ EM_002, WP5_ EM_003, WP5_ EM_011
Related use cases	Use case "Specify Traceability between Managed Artefacts, Conduct Impact Analysis of Managed-Artefact Change"
Input	• Assurance project • Artefact Model with at least two Artefacts, of the Assurance Project
Steps	1. Open the Artefact Model 2. Select an Artefact

	3. Add an Artefact Rel to the Artefact 4. Select the Artefact as source 5. Indicate 'Modify' as Change Effect Kinds 6. Select another Artefact as target 7. Save the Artefact Model 8. Add a Modification Event to the first Artefact (i.e. Artefact Rel source) 9. Save the Artefact Model
Expected results	It has been possible to execute all the steps, and impact analysis is triggered
Priority	Must

Table 29. Defined Test Case WP5_TC_04 for WP5_5.10, WP5_5.13, WP5_5.21 functionalities

ID	WP5_TC_04
Scope	Specification of process information for artefacts
Functionality ID	WP5_ EM_010, WP5_ EM_013
Related use cases	Use case "Specify Executed-Process Information for Artefact Use"
Input	• Assurance project • Artefact Model with Artefacts, of the Assurance Project
Steps	1. Open the Assurance Project 2. Create a Process Model for the Assurance Project 3. Add an Activity 4. Fill the fields of the Activity 5. Add an Organization 6. Fill the fields of the Organization 7. Add a Technique 8. Fill the fields of the Technique 9. Associate the Technique to the Activity 10. Add a Person 11. Fill the fields of the Person 12. Associate the Person to the Activity 13. Associate the Person to the Organization 14. Select some Artefacts as required Artefacts for the Activity 15. Select some Artefacts as produced Artefacts for the Activity 16. Add a sub-Activity to the Activity 17. Select an Artefacts as owned Artefact for the Person 18. Select an Artefacts as created Artefact for the Technique 19. Add another Activity 20. Associate the latter Activity with the first one
Expected results	It has been possible to execute all the steps, and all the information specified for the Process Model is available
Priority	Must

6.3 Evidence Management Test Results

Table 30 presents, for each test case defined for the implemented Evidence Management basic building block functionalities, the results of the execution, the status, a rationale when the execution failed and the AMASS project partner who is responsible for the validation of the test case. The tools used to perform the test cases execution are a PostgreSQL server, the OpenCert software version in a development version, and a local CDO repository. The installation instructions for these tools are provided in the AMASS Developer

Guide [9]. The validation data used to run the test cases have been restored from a databased backup[8] provided by the responsible project partner. The AMASS User Manual [8] was used to understand how the Evidence Management functionalities provided for Prototype Core were working.

The test cases have been performed with the following machine configuration: Operating system: Windows 10 Enterprise (64 bits), Processor: Intel Core i7-6500U, CPU @ 2.50 GHz, RAM: 8 GB. All the test cases have successfully been passed for the building block.

Table 30. Test results for the implemented Evidence Management functionalities

Test Case ID	Execution Results	Status	Rationale	Responsible
WP5_TC_01	An artefact model with the specified information	Passed		TRC
WP5_TC_02	An artefact model with the specified information	Passed		TRC
WP5_TC_03	Impact analysis is triggered	Passed		TRC
WP5_TC_04	A process model with the specified information	Passed		TRC

[8] The database backup used is OpenCert1, located in the SVN repository https://services-medini.kpit.com/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/Vaditation_Data/OpenCert

7. Testing and Validation for Compliance Management Basic Building Block

7.1 Compliance Management Functionalities for Prototype Core

The functionalities concerning Compliance Management basic building block are defined in the D2.1 deliverable [10]. Table 31 is an excerpt of these functionalities planned for Prototype Core, their implementation status and the AMASS project partner who is responsible for the validation of the test case. Among the five planned functionalities for Prototype Core, one functionality was not implemented and postponed for the next version of the AMASS platform.

Table 31. Compliance Management basic building block functionalities

ID	Functionality	Status	Responsible
WP6_CM_001	Retrieving, digitalizing and storing of a set of industrial standards (including the parts, objectives, practices, goals/requirements, criticality levels from the standards)	Implemented	TEC
WP6_CM_002	Specification of the interpretation of how to comply with an industrial standard in a specific project (e.g., check list with specific compliance requirements)	Implemented	TEC
WP6_CM_005	Web-based monitoring of Compliance status to be filtered by any custom criteria.	Implemented	TEC
WP6_CM_008	The AMASS tools shall enable users to visualize process compliance. This could be done via compliance maps (matrix) or via arguments aimed at justifying the satisfaction of the requirements coming from the standards.	Implemented	TEC
WP6_CM_006	Compliance status to externals	Postponed	

7.2 Compliance Management Test Cases

The tables in this section below define the test cases to validate the correct implementation of the Compliance Management basic building block of Prototype Core. Test cases have been defined based on the use case scenarios defined in the D2.2 deliverable for the concerned functionalities when existing.

Table 32. Defined Test Case WP6_TC_01 for WP6_6.1 functionality

ID	WP6_TC_01
Scope	Retrieve, digitalize and store a set of norms, recommendations, standards, or quality models.
Functionality ID	WP6_CM_001
Related use cases	Use case “Capture information from standards”
Input	Standard information
Steps	1. Create a new standard model 2. Specify the characteristics that define the standard in the properties view 3. Structure/Categorize the standard by parts, objectives, activities, practices, goals and requirements 4. Describe the parts, objectives, activities, practices, goals and requirements contained in the standard in the properties view

Expected results	Standard model
Priority	Must

Table 33. Defined Test Case WP6_TC_02 for WP6_6.1 functionality

ID	WP6_TC_02
Scope	Retrieve, digitalize and store a set of norms, recommendations, standards, or quality models.
Functionality ID	WP6_CM_001
Related use cases	Use case “Capture information from standards”
Input	RefFramework in OpenCert
Steps	1. Create a project baseline from a standard model -
Expected results	Project Baseline
Priority	Must

Table 34. Defined Test Case WP6_TC_03 for WP6_6.1 functionality

ID	WP6_TC_03
Scope	Retrieve, digitalize and store a set of norms, recommendations, standards, or quality models.
Functionality ID	WP6_CM_001
Related use cases	Use case “Capture information from standards”
Input	Baseline in OpenCert
Steps	1. Generate argument fragments for the assurance case in relation with process-based argumentation from the baseline
Expected results	Argument fragments
Priority	Must

Table 35. Defined Test Case WP6_TC_04 for WP6_6.2 functionality

ID	WP6_TC_04
Scope	Create, modify and drop assurance information
Functionality ID	WP6_CM_002
Related use cases	Use case “Manage Assurance Project”
Input	Library and configuration models exported from EPF
Steps	1. Import process related information from EPF
Expected results	Process and Artefact (Evidence) models
Priority	Must

Table 36. Defined Test Case WP6_TC_05 for WP6_6.2, WP6_6.6 functionalities

ID	WP6_TC_05
Scope	Create, modify and drop assurance information
Functionality ID	WP6_CM_002, WP6_CM_008
Related use cases	Use case “Manage Assurance Project”
Input	A model containing information of the standard available in the platform
Steps	1. Create a new assurance project 2. Specify the baseline in association with a standard which will be followed in the project 3. Specify the compliance maps/links through the project lifecycle.
Expected results	Assurance Project
Priority	Must

Table 37. Defined Test Case WP6_TC_06 for WP6_6.3 functionality

ID	WP6_TC_06
Scope	Information about the assurance activities
Functionality ID	WP6_CM_005
Related use cases	Use case “Monitor Assurance Project Status”
Input	Assurance project in the platform
Steps	1. Select an assurance project 2. Define a filter to find specific compliance information
Expected results	Compliance information
Priority	Must

7.3 Compliance Management Test Results

Table 38 presents, for each test case defined for the implemented Compliance Management basic building block functionalities, the results of the execution, the status, a rationale when the execution was not fully satisfying the expected results, and the AMASS project partner who is responsible for the validation of the test case.

The installation instructions for the validation environment and the description of the Compliance Management functionalities provided for Prototype Core are respectively found in the AMASS Developer Guide [9] and the AMASS User Manual [8]. As testing data, we use a database backup containing examples of assurance project and evidence model. We also used some files exported from EPF tool for the process model[9].

The test cases have been performed with the following machine configuration: Operating system: Windows 7 Enterprise (64 bits), Processor: Intel Core i7-56000U, CPU @ 2.60 GHz, RAM: 16 GB. Four test cases have successfully been passed for the building block, when two test cases have not been fully corresponding to the expected results.

Table 38. Test results for the implemented Compliance Management basic building block functionalities

Test Case ID	Execution Results	Status	Rationale	Responsible
WP6_TC_01	A standard’s model with its characteristics	Passed but	There is no practices and goals to fill in in the properties view for the element	CEA
WP6_TC_02	A new baseline model	Passed		CEA
WP6_TC_03	An argumentation model	Passed but	The argumentation model is not generated from the baseline, but together with the baseline from a new assurance project creation. When a different baseline is created, we are not able to generate the corresponding argumentation model	CEA
WP6_TC_04	Process and artefact models	Passed		CEA

[9] The EPF files used are located in the SVN repository https://services-medini.kpit.com/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/Vaditation_Data/EPF/Exported XML

	imported from EPF			
WP6_TC_05	An Assurance Project with compliance links done. Summary can be checked through the mapping table	Passed		CEA
WP6_TC_06	The Compliance information related to a specific element type (activity, requirement, etc.)	Passed		CEA

8. Prototype Core Validation Synthesis

8.1 Analysis of Test Results

Table 39 summarizes the implementation status of the Prototype Core functionalities per basic building block. In total, 43 functionalities have been planned for the Core Prototype: nine of them have been postponed, while 30 functionalities have successfully been implemented and four functionalities have not met the expected behaviours.

Table 39. Prototype Core Implementation Status

Functionalities	System Component Specification	Assurance Case Specification	Evidence Management	Compliance Management	AMASS Prototype Core
Correctly Implemented	14	3	9	4	30
Implemented but required amelioration	0	3	0	1	4
Postponed	2	1	5	1	9
Total	**16**	**7**	**14**	**6**	**43**

We defined 32 test cases to test and validate the (34) implemented functionalities of AMASS Prototype Core: 27/32 test cases have been successfully PASSED including all the ones defined for System Component Specification and Evidence Management basic building blocks. Two test cases result with the status PASSED BUT, both concerning Compliance Management functionalities. Finally, three test cases concerning Assurance Case Specification functionalities FAILED to provide the expected results. For each test case with the status FAILED or PASSED BUT, we created a ticket corresponding to the problem identified in the software or user guide (user manual and developer guide) in the AMASS wiki to report them to the implementation responsible.

Table 40. Results of the test cases for Prototype Core implemented functionalities

Test Results Status	System Component Specification	Assurance Case Specification	Evidence Management	Compliance Management	AMASS Prototype Core
Passed	15	3	4	5	27
Passed but	0	0	0	2	2
Failed	0	3	0	0	3
Total	**15**	**6**	**4**	**7**	**32**

From these results, we have the following findings:

- Difficulty to have consistent and up to date reference documents to base the tests on. The D2.1 deliverable, the D2.2 deliverable and the user manual, which are the main input documents for the tests, were in-progress when we started the validation task. Hence, the tests are based on requirements and use case scenarios that were either incomplete or no more up to date, making it difficult to exploit. We also found some discrepancies between the User Manual and Developer Guide and the software in several cases: either they were not as up to date as the software or vice versa.
- Some required functionalities have not been correctly or not completely implemented. In some cases, the platform did not implement some functionalities that were required for the AMASS Prototype Core (postponed functionalities). In other cases, the expected results of the test cases have not met because

the relative feature is simply not implemented, e.g. regarding the vocabulary and the argumentation model for the Assurance Case Specification basic building block.

- The tests execution on the platform revealed some problems concerning the tool integration between the basic building blocks as well as some usability and performance concerns.

8.2 Recommendations

In this section, we make some recommendations considering the results of testing and validation of the AMASS Prototype Core:

- In priority, we must correct bugs with respect to new developments and implement the postponed functionalities. This is because, firstly, the cost of a bug correction and tests increases with time, and secondly, some functionalities targeted in the version 2 of the AMASS platform (Prototype P1) will be built on top of the ones planned in this Prototype Core.
  - ➔ Recommendation: Use an issue-tracker system to follow the status of bug report.
- We must update the user manual and developer guide with the requirements and the use case scenarios defined in the deliverables D2.1 and D2.2 in order to create a better alignment between these documents. To this end,
  - it will be useful to define the means to facilitate the traceability analysis between
    - requirements for the AMASS platform,
    - the functionalities defined (at a conceptual level) and implemented to meet them,
    - the test cases and results resulting from the validation.
  - ➔ Recommendation: Report any inconsistency between requirements and functionalities in the issue-tracking system. Make sure these inconsistencies are solved before the next validation iteration. Ideally, define a traceability matrix to assist in checking the completeness and consistency of relationships between requirements, conceptual models, design, test cases and test results.
  - the requirements and use cases may also be written in a more homogeneous manner to ensure internal and external consistency.
    - ➔ Recommendation: Define strict template and vocabulary for requirement and use case definition.
- In addition to the user manual, it would also be very helpful to have ready methodological guidelines to understand and use the platform, since the user manual specifies how to perform an action or launch a functionality but does not include the overall rationale for performing such actions and functionalities (in this way).
  - ➔ Recommendation: Define methodological guidelines for AMASS platform.
- The next AMASS Prototype P1 will extend the basic building blocks of the Prototype Core with four (4) pillars which correspond to specific project Scientific and Technical Objectives (STO), namely : Architecture-Driven Assurance (STO1), Multi-concern Assurance (STO2), Seamless Interoperability (STO3) and Cross/Intra-Domain Reuse (STO4) (see Figure 1). The validation of the AMASS Prototype P1 will then focus on integration and interoperability testing in addition to the validation of the individual components.
  - ➔ Recommendation: Define specific test cases to validate the integration of the individual components of the platform.
- To enhance further the validation results, the test cases definition by the validation team must be carried out in closer collaboration with the implementation team prior to their execution, to early identify any comprehension discrepancies of the implemented functionalities.
  - ➔ Recommendation: Define a test cases review and validation phase before their execution.

Abbreviations and Definitions

AMASS	Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems
API	Application Programming Interface
ARTA	AMASS Reference Tool Architecture
ATL	ATLAS Transformation Language
AUTOSAR	AUTomotive Open System ARchitecture
BPMN	Business Process Model and Notation
BSCU	Braking System Control Unit
BVR	Base Variability Resolution
CACM	Common Assurance and Certification Meta-model
CBSE	Component-Based Software Engineering
CCL	Common Certification Language
CDO	Connected Data Objects
COTS	Commercial Off The Shelf
CPS	Cyber Physical Systems
CPU	Central Processing Unit
CVL	Common Variability Language
DSL	Domain Specific Language
EPF	Eclipse Process Framework
GB	Gigabyte
GSN	Goal Structuring Notation
IMA	Integrated Modular Avionics
MDE	Model Driven Engineering
MOF	Meta Object Facility
MOTS	Modified off the shelf
OMG	Object Management Group
OSLC	Open Services for Lifecycle Collaboration
OTS	Off the shelf
PLE	Process Line Engineering
RAM	Random-access memory
RQS	Requirements Quality Suite
SACM	Structured Assurance Case Meta-model
SEI	Software Engineering Institute
SEooC	Safety Element out of Context
SKR	System Knowledge Repository
SOUP	Software of Unknown Pedigree
SPEM	Software and Systems Process Engineering Meta-model
STO	Scientific and Technical Objectives
SVN	Subversion
TRL	Technology Readyness Level

UDP	User-defined Process
V&V	Verification & Validation

References

[1] OPENCOSS project. 2015. http://www.opencoss-project.eu

[2] SafeCer Project. 2015. http://safecer.eu

[3] CHESS project. 2015. http://www.chess-project.org/

[4] PolarSys. https://www.polarsys.org

[5] PolarSys: CHESS project. https://www.polarsys.org/projects/polarsys.chess

[6] PolarSys: OpenCert project. https://www.polarsys.org/projects/polarsys.opencert

[7] Eclipse Process Framework Project (EPF) https://eclipse.org/epf/

[8] AMASS project: Prototype Core User Manual[10]. 2017. https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/AMASS_Prototype1_UserManual.docx

[9] AMASS Developer Guide[11]. 2017 https://services-medini.kpit.com/AMASS/browser/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeCore/AMASS_Prototype1_DeveloperGuide.doc

[10] AMASS D2.1 Business cases and high-level requirements. 28 February 2017.

[11] AMASS D2.2 AMASS reference architecture (a). 30 November 2016.

[12] AMASS D3.4 - Prototype for architecture-driven assurance (a). 23 December 2016.

[13] AMASS D4.4 - Prototype for multiconcern assurance (a). 31 January 2017.

[14] AMASS D5.4 - Prototype for seamless interoperability (a). 31 March 2017.

[15] AMASS D6.4 - Prototype for cross/intra-domain reuse (a). 31 March 2017.

[10] The current User Manual is a draft document; the final version of the manual will be integrated in D2.5 - AMASS User guidance and methodological framework (m31).

[11] The current Developer Guide is a draft document; the final version of the manual will be integrated in D2.5 - AMASS User guidance and methodological framework (m31).

Appendix A: Validation status of the basic building blocks

The table below summarizes the validation status of the basic building blocks. Each functionality of the Prototype Core is traced to the test cases that evaluated them, if existing. The colour code indicates that the status of associated test cases to the functionalities:

- Green indicates that all test cases have the status PASSED hence the functionality is correctly implemented.
- Red indicates that the test cases FAILED hence the functionality was not correctly implemented.
- Orange indicates the test cases PASSED BUT we identified some needed improvements to fully meet the expected results for the functionality.
- Yellow indicates that we did not define test cases to evaluate the functionality, as it is postponed for next version of the platform, so not implemented yet.

Table 41. Prototype Core Functionalities Status

Functionality		Test Cases Status
System components specification		
WP3_SC_001	Browse along the different abstractions levels	WP3_TC_01
WP3_SC_002	Move, edit along the different abstractions levels	WP3_TC_02
WP3_SC_004	Formalize requirements with formal properties	WP3_TC_14
WP3_SC_005	Allocating requirements to parts of the component model.	WP3_TC_16
WP3_SC_006	Specify the component behavioural model	WP3_TC_19
WP3_SC_007	Fault injection (include faulty behaviour of a component)	
WP3_SAM_001	Trace all the assurance information with the specific component	WP3_TC_03
WP6_RA_003	Reuse of pre-developed components and artefacts	WP3_TC_25
WP3_VVA_001	Trace contract evidence and assurance case	WP3_TC_06
WP3_VVA_004	Trace requirements validation checks	
WP3_CAC_002	Associate a contract to a component	WP3_TC_30
WP3_CAC_002	Drop contract from component	WP3_TC_31, 32
WP3_CAC_003	Structure properties into contracts (assumptions/guarantees)	WP3_TC_15
WP3_CAC_004	Refinement of a contract along the hierarchical components	WP3_TC_29
WP3_CAC_012	Browse Contract status	WP3_TC_33
WP3_CAC_013	Specify contracts : the assumption and the guarantee elements	WP3_TC_05
Assurance Case Specification		
WP4_ACS_001	Edit an assurance case in a scalable way	WP4_TC_01
WP4_ACS_003	Instantiate in the actual assurance case an argument pattern	WP4_TC_05
WP4_ACS_004	semi-automatic generation of process arguments	
WP4_ACS_005	Provide support for language formalization in argument claims	WP4_TC_06
WP4_ACS_010	Capability of generating a compositional assurance case arg.	WP4_TC_03
WP4_4.8	Navigation from an evidence supporting a claim to its information	WP4_TC_04
WP4_4.15	Edit an assurance case in collaboration with other people	
WP4_4.19	Edit and store argumentation patterns for later use.	WP4_TC_08
Evidence Management		
WP5_EM_001	Evidence characteristics specification	WP5_TC_01
WP5_EM_002	Evidence traceability	WP5_TC_03
WP5_EM_003	Evidence change impact analysis	WP5_TC_03
WP5_EM_004	Evidence evaluation	WP5_TC_02
WP5_EM_005	Evidence information import	WP5_TC_01
WP5_EM_006	Evidence information export	
WP5_EM_008	Visualization of chains of evidence	
WP5_EM_009	Suggestion of evidence traces	
WP5_EM_010	Evidence lifecycle information storage	WP5_TC_02, 04
WP5_EM_011	Interactive evidence change impact analysis	WP5_TC_03

WP5_EM_012	Evidence trace verification	
WP5_EM_013	Link of evidence to other assets	WP5_TC_04
WP5_EM_014	Evidence resource specification	WP5_TC_01
WP5_EM_015	Resource part selection	
Compliance management		
WP6_CM_001	Retrieving, digitalizing and storing of a set of industrial standards	WP6_TC_01, 02, 03
WP6_CM_002	Specify how to comply with an industrial standard	WP6_TC_04, 05, 07
WP6_CM_005	Monitoring of Compliance status, filtering by custom criteria.	WP6_TC_06
WP6_CM_006	Compliance status to externals	
WP6_CM_008	Visualize of process compliance via maps or arguments	WP6_TC_05