FutureTrust

Implementation and Test Plan

D.4.1

Document Identification	
Date	25/04/2017
Status	Final
Version	1.02

Related WP	Implementation and Test Plan	**Document Reference**	D4.1
Related Deliverable(s)	Insert Related Deliverables	**Dissemination Level**	PU/CO
Lead Participant	ARHS	**Lead Author**	Vincent Bouckaert (ARHS) Melis Özgür Çetinkaya Demir (TUBITAK)
Contributors	Muhammet Yıldız	**Reviewers**	Alexandre Defays (ARHS) Edona Fasllija (TUBITAK) Jens Urmann (G+D) Juraj Somorovsky (RUB) Detlef Hühnlein (ECSEC)

Abstract: The present document provides an overview of the planning defined for the development of the various components that are part of the Future Trust project. It also specifies the standards and methodologies used for software development and quality assurance, along with descriptions of the development and hosting environments that will be deployed by the involved stakeholders. The test plan section describes the scope, methodology, required resources and schedule of the testing activities among the FutureTrust project. The Master Test Plan provides the overall test planning and

unifies other level specific test plans like Unit Test Plan, Integration Test Plan, System Test Plan. The structure of this document is based on (ISO/IEC/IEEE 29119, 2013).

1. Executive Summary

This document provides a description of how the services and pilots designed during Work Package 3 will be implemented, tested and deployed.

The implementation plan contains an overview of the overall eco-system being developed during the FutureTrust project along with a description of the various tasks involved in its implementation.

This document also specifies the standards and methodologies that will be used for the software development activities as well as for quality assurance and testing.

The test plan section describes the scope, methodology, required resources and schedule of the testing activities among the FutureTrust project. The Master Test Plan provides the overall test planning and unifies other level specific test plans like Unit Test Plan, Integration Test Plan, System Test Plan. The structure of this document is based on (ISO/IEC/IEEE 29119, 2013).

2. Document Information

2.1 Contributors

Name	Partner
Vincent Bouckaert	ARHS
Alexandre Defays	ARHS
Melis Özgür Çetinkaya Demir	TUBITAK
Muhammet Yıldız	TUBITAK

2.2 History

Version	Date	Author	Changes
0.1	23/03/2017	Vincent Bouckaert	Initial draft
0.2	07/04/2017	Melis Özgür Çetinkaya Demir	Intro for Test Plan, Unit Test Plan, Integration Test Plan, System Integration Test Plan, Appendix
0.3	07/04/2017	Muhammet Yıldız	System Integration Test Plan
0.4	07/04/2017	Edona Fasllija	Review for Test Plan section
0.5	12/04/2017	Vincent Bouckaert	General review, update based on comments provided by G+D and Tubitak
0.6	14/04/2017	Melis Özgür Cetinkaya Demir	General review
1.0	18/04/2017	Vincent Bouckaert	Merging comments and modifications provided by G+D and Tubitak, moving version number to 1.0
1.01	25/04/2017	Vincent Bouckaert	Minor update taking into account comments provided by Multicert.
1.02	01/05/2017	Vincent Bouckaert	Update taking into account comments provided by RUB, ECSEC and PSDA.

2.3 Table of Contents

2.4 Table of Figures

2.5 Table of Tables

2.6 Table of Acronyms

Acronym	Meaning
App	Application
ASiC	Associated Signature Container
CA	Certification Authority
CAdES	CMS Advanced Electronic Signature
CMS	Cryptographic Message Syntax

Acronym	Meaning
CO	Dissemination Level "Confidential"
CR	Change Request
DesignTeam	Design Team of Future Trust components
DevTeam	Development Team of Future Trust components
EN	European Norm
eID	Electronic Identification
eIDAS	Electronic Identification and Authentication
ETSI	European Telecommunications Standards Institute
EU	European Union
FAT	Factory Acceptance Test
GUI	Graphical User Interface
HSM	Hardware Security Module
IdMS	Identity Management Service
mSignS	Mobile Signature Service
MTDL	Minder Test Definition Language
PAdES	PDF Advanced Electronic Signature
PDF	Portable Document Format
R*x*	Requirement
SUT	System Under Test
TA	Test Assertion
TDes	Test Designer
TD	Test Developer
TestTeam	Test Team of Future Trust project
TL	Trusted List
TLSO	Trusted List Scheme Operator
TS	Technical Specification
TSP	Trust Service Provider
TSL	Trust-service Status List
UML	Unified Modelling Language
XAdES	XML Advanced Electronic Signature
XML	Extensible Markup Language

3. Project Description

Against the background of the regulation 2014/910/EU on electronic identification (eID) and trusted services for electronic transactions in the internal market (eIDAS), the FutureTrust project, which is funded within the EU Framework Programme for Research and Innovation (Horizon 2020) under Grant Agreement No. 700542, aims at supporting the practical implementation of the regulation in Europe and beyond.

For this purpose, the FutureTrust project will address the need for globally interoperable solutions through basic research with respect to the foundations of trust and trustworthiness, actively support the standardisation process in relevant areas, and provide Open Source software components and trustworthy services which will ease the use of eID and electronic signature technology in real world applications. In particular, the FutureTrust project will extend the existing European Trust Service Status List (TSL) infrastructure towards a "Global Trust List", develop a comprehensive Open Source Validation Service as well as a scalable Preservation Service for electronic signatures and seals and will provide components for the eID-based application for qualified certificates across borders, and for the trustworthy creation of remote signatures and seals in a mobile environment.

4. Implementation

4.1 Components

The eco-system foreseen in the context of FutureTrust, illustrated in Figure 1, is composed of the following components:

- A Global Trust Service Status List (gTSL), extending the current trust management schemes to support eID providers while also enabling the inclusion of trust service providers from outside of the European Union;
- A Comprehensive Validation Service (ValS), exposing the functionalities necessary for the validation of Advanced Electronic Signatures (AdES) as well as other cryptographic artefacts such as authentication tokens, evidence records, etc.;
- A Scalable Preservation Service (PresS), providing the means to preserve the conclusiveness of electronic signatures and other signed data over prolonged periods of time;
- Identity Management Services (IdMS), providing a flexible and secure mobile authentication solution on a selected mobile platform, that can be used for remote signing and sealing services as well as other server-based trusted services;
- A Remote Signing and Sealing service (mSignS).

Also, in order to showcase these various services, several pilot applications will also be implemented:

- An e-Invoice pilot portal (Austria);
- An e-Mandates demonstrator application (Portugal);
- An e-Apostille demonstrator application (Georgia);
- A BVA demonstrator application (Germany).

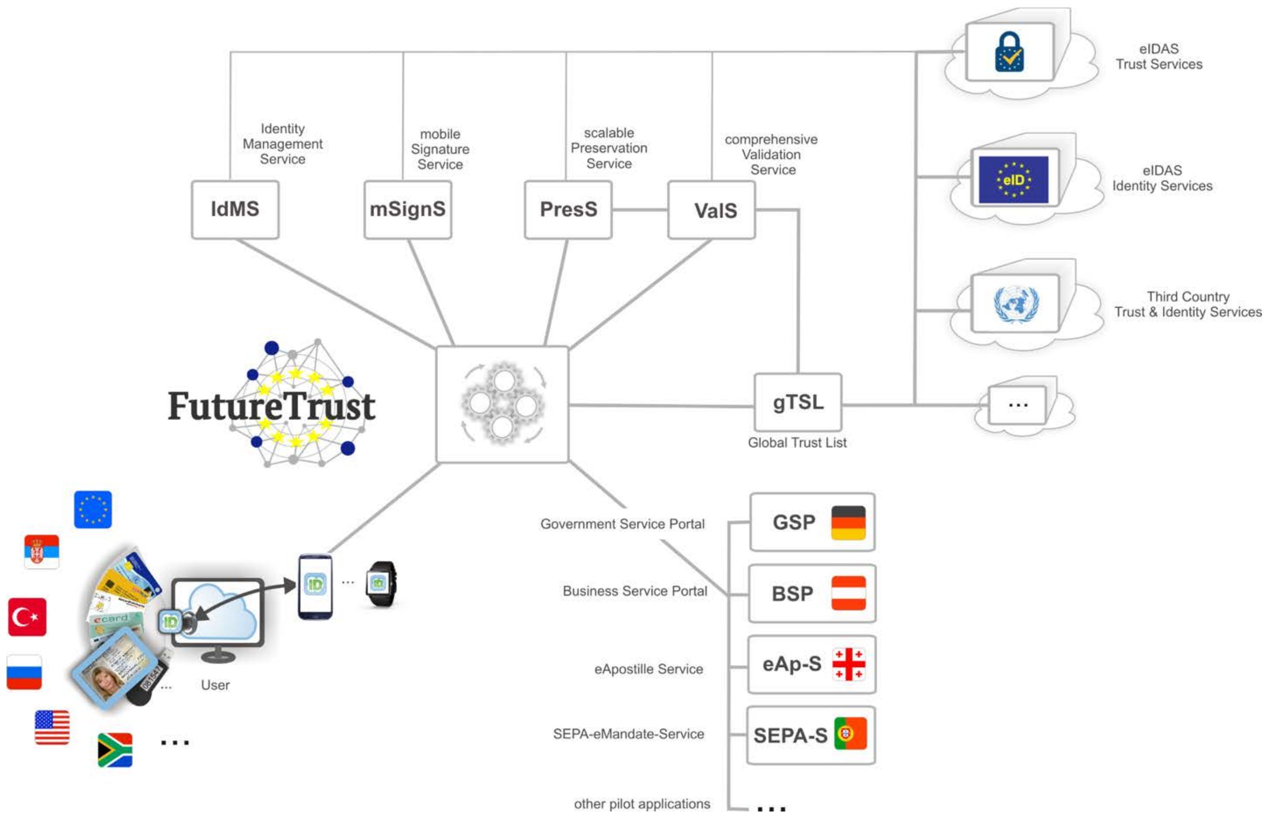


Figure 1 - Future Trust eco-system

4.2 Schedule

The following section provides indicative schedules for the implementation of the FutureTrust services and pilots and their alignment planning.

4.2.1 Services implementation

An overview of the deliverables is provided in Table 1, including the related design deliverables for each service and the contractual end dates set for each of these components.

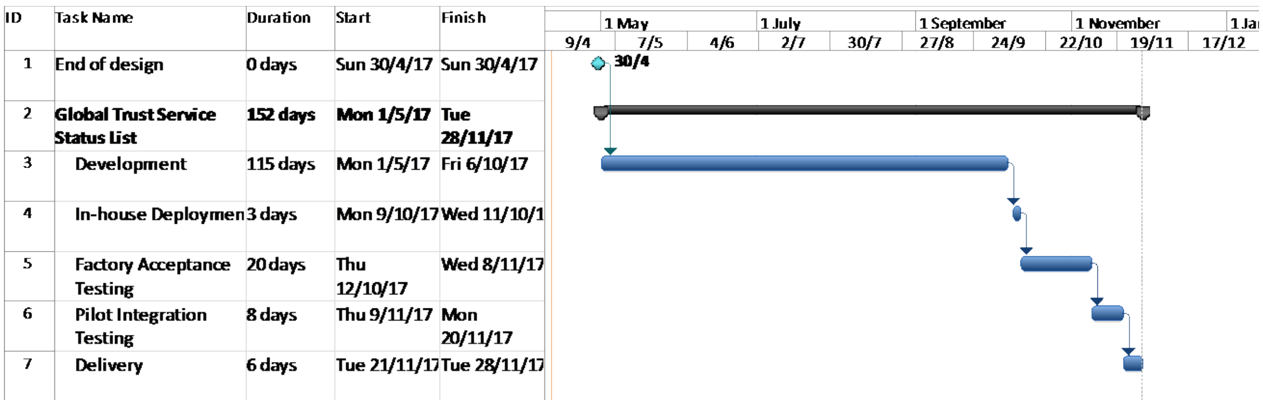
Deliverable ID	Deliverable title	Contractual end date	Related Design Deliverable	Contractual end date
D4.2	Global Trust Service Status List	Nov-17	D3.2	May-17
D4.4	Comprehensive Validation Service	May-18	D3.3	May-17
D4.5	Scalable Preservation Service	Aug-18	D3.4	May-17
D4.6	Identity Management Service	Aug-18	D3.5	May-17
D4.7	Remote Signing and Sealing	Aug-18	D3.6	May-17

Table 1 - Future Trust services: Overview of deliverables

The dates provided in the below sections are indicative only and might be refined during execution of Work Package 4.

4.2.1.1 Global Trust Service Status List

The implementation planning foreseen for the Global Trust Service Status List component is provided as a simplified Gantt chart in Figure 2.

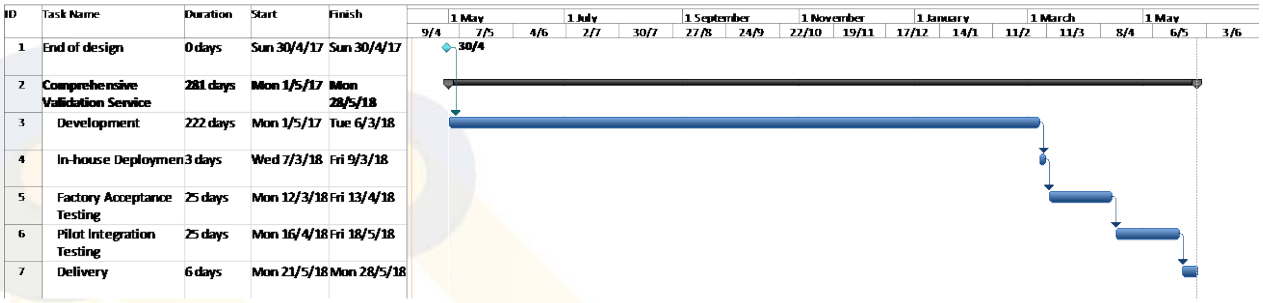


ID	Task Name	Duration	Start	Finish
1	End of design	0 days	Sun 30/4/17	Sun 30/4/17
2	**Global Trust Service Status List**	**152 days**	**Mon 1/5/17**	**Tue 28/11/17**
3	Development	115 days	Mon 1/5/17	Fri 6/10/17
4	In-house Deploymen	3 days	Mon 9/10/17	Wed 11/10/1
5	Factory Acceptance Testing	20 days	Thu 12/10/17	Wed 8/11/17
6	Pilot Integration Testing	8 days	Thu 9/11/17	Mon 20/11/17
7	Delivery	6 days	Tue 21/11/17	Tue 28/11/17

Figure 2 - Global Trust Service Status List: Indicative Gantt chart

4.2.1.2 Comprehensive Validation Service

The implementation planning foreseen for the Comprehensive Validation Service component is provided as a simplified Gantt chart in Figure 3.

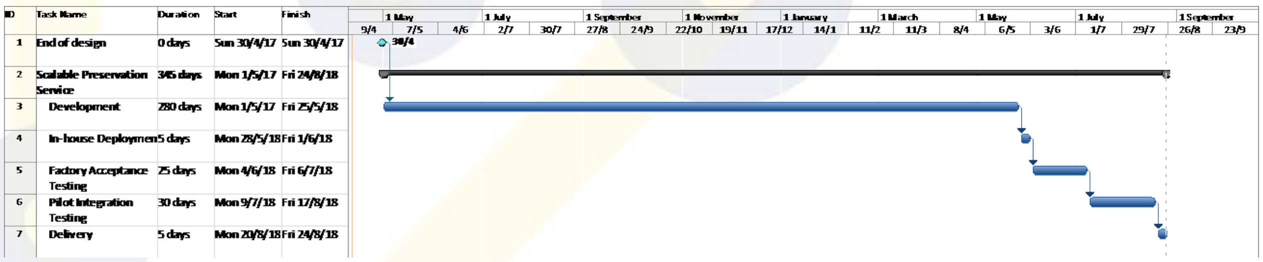


ID	Task Name	Duration	Start	Finish
1	End of design	0 days	Sun 30/4/17	Sun 30/4/17
2	**Comprehensive Validation Service**	**281 days**	**Mon 1/5/17**	**Mon 28/5/18**
3	Development	222 days	Mon 1/5/17	Tue 6/3/18
4	In-house Deploymen	3 days	Wed 7/3/18	Fri 9/3/18
5	Factory Acceptance Testing	25 days	Mon 12/3/18	Fri 13/4/18
6	Pilot Integration Testing	25 days	Mon 16/4/18	Fri 18/5/18
7	Delivery	6 days	Mon 21/5/18	Mon 28/5/18

Figure 3 - Comprehensive Validation Service: Indicative Gantt chart

4.2.1.3 Scalable Preservation Service

The implementation planning foreseen for the Scalable Preservation Service component is provided as a simplified Gantt chart in Figure 4.

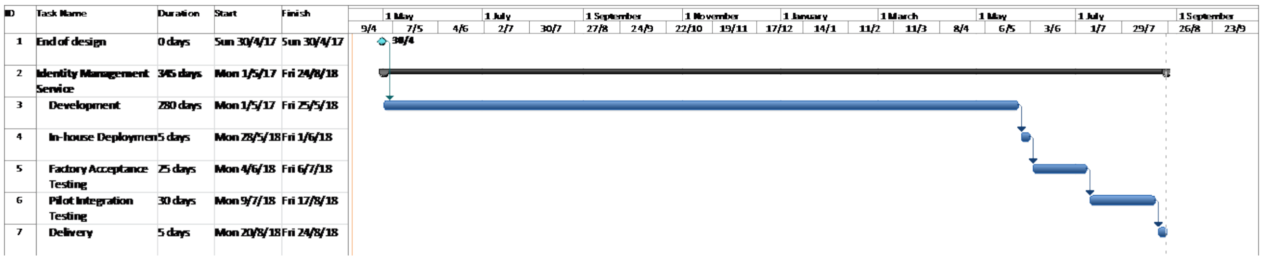


ID	Task Name	Duration	Start	Finish
1	End of design	0 days	Sun 30/4/17	Sun 30/4/17
2	**Scalable Preservation Service**	**345 days**	**Mon 1/5/17**	**Fri 24/8/18**
3	Development	280 days	Mon 1/5/17	Fri 25/5/18
4	In-house Deploymen	5 days	Mon 28/5/18	Fri 1/6/18
5	Factory Acceptance Testing	25 days	Mon 4/6/18	Fri 6/7/18
6	Pilot Integration Testing	30 days	Mon 9/7/18	Fri 17/8/18
7	Delivery	5 days	Mon 20/8/18	Fri 24/8/18

Figure 4 - Scalable Preservation Service: Indicative Gantt chart

4.2.1.4 Identity Management Service

The implementation planning foreseen for the Identity Management Service component is provided as a simplified Gantt chart in Figure 5.

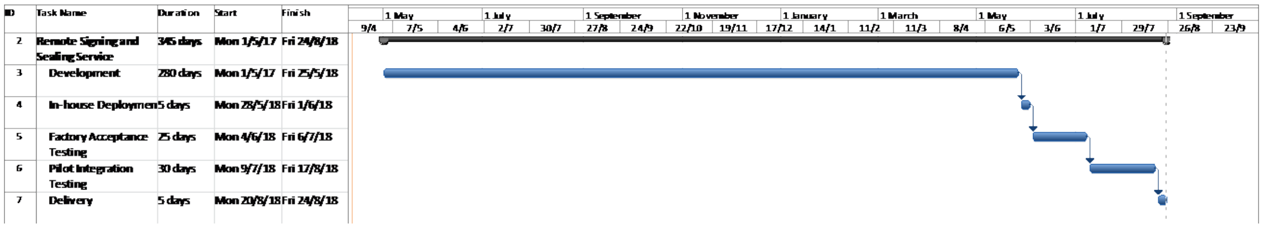


ID	Task Name	Duration	Start	Finish
1	End of design	0 days	Sun 30/4/17	Sun 30/4/17
2	Identity Management Service	345 days	Mon 1/5/17	Fri 24/8/18
3	Development	280 days	Mon 1/5/17	Fri 25/5/18
4	In-house Deploymen	5 days	Mon 28/5/18	Fri 1/6/18
5	Factory Acceptance Testing	25 days	Mon 4/6/18	Fri 6/7/18
6	Pilot Integration Testing	30 days	Mon 9/7/18	Fri 17/8/18
7	Delivery	5 days	Mon 20/8/18	Fri 24/8/18

Figure 5 - Identity Management Service: Indicative Gantt chart

4.2.1.5 Remote Signing and Sealing Service

The implementation planning foreseen for the Remote Signing and Sealing Service component is provided as a simplified Gantt chart in Figure 6.



ID	Task Name	Duration	Start	Finish
2	Remote Signing and Sealing Service	345 days	Mon 1/5/17	Fri 24/8/18
3	Development	280 days	Mon 1/5/17	Fri 25/5/18
4	In-house Deploymen	5 days	Mon 28/5/18	Fri 1/6/18
5	Factory Acceptance Testing	25 days	Mon 4/6/18	Fri 6/7/18
6	Pilot Integration Testing	30 days	Mon 9/7/18	Fri 17/8/18
7	Delivery	5 days	Mon 20/8/18	Fri 24/8/18

Figure 6 - Remote Signing and Sealing Service: Indicative Gantt chart

4.2.2 Pilot applications

The following section provides the schedule for the implementation of the Future Trust pilots. An overview of the deliverables is provided in Table 2, including the related design deliverables for each pilot.

Deliverable ID	Deliverable title	Contractual end date	Related Design Deliverable	Contractual end date
D4.8	e-Mandates Demonstrator Implementation Documentation	Feb-19	D3.7	May-17
D4.9	e-Invoice Pilot Implementation Documentation	Feb-19	D3.7	May-17
D4.10	BVA Demonstrator Implementation Documentation	Feb-19	D3.7	May-17
D4.11	eApostille Demonstrator Implementation Documentation	Feb-19	D3.7	May-17

Table 2 - Pilot applications: Overview of deliverables

4.2.3 Alignment between Future Trust services and pilot applications

The following section provides a description of the links between each FutureTrust service and its related pilot applications, along with an alignment planning for these.

Since there are no final specifications for the pilot applications at the time of writing, the use case scenarios listed below may be complemented in a subsequent phase.

4.2.3.1 Global Trust Service Status List

The Global Trust Service Status List is the sole component of the Future Trust project that is not directly linked to a pilot, as it is only used by the Comprehensive Validation Service (although other users can contact it to retrieve trust status information in an ad-hoc manner).

4.2.3.2 Comprehensive Validation Service

Table 3 provides an overview of the pilots that rely on the Comprehensive Validation Service, along with a description of the type of cryptographic artefacts that are expected to be validated and the scenario in which this validation should occur.

Pilot	Cryptographic artefact type	Use case scenario
e-Mandates	Electronic signature embedded in SEPA e-Mandates	Signature validation
eApostille	Electronic signature embedded in e-Apostille documents	Signature validation
e-Invoice	Electronic signature embedded in e-Invoices	Signature validation
	Authentication tokens	User authentication

Table 3 - Comprehensive Validation Service: links to pilots

4.2.3.3 Scalable Preservation Service

Table 4 provides an overview of the pilots that rely on the Scalable Preservation Service, along with a description of the scenarios in which this service will be involved.

Pilot	Use case scenario
e-Mandates	e-Mandate long-time preservation
eApostille	Long-time preservation of e-Apostille documents
e-Invoice	No specific scenario at this time

Table 4 - Scalable Preservation Service: links to pilots

4.2.3.4 Identity Management Service

Table 5 provides an overview of the pilots that rely on the Identity Management Service, along with a description of the scenario in which this service will be involved.

Pilot	Use case scenario
e-Mandates	Debtor (user) authentication on Debtor's Bank (homebanking)
eApostille	User authentication for long-term preservation

Table 5 - Identity Management Service: links to pilots

4.2.3.5 Remote Signing and Sealing

Table 6 provides an overview of the pilots that rely on the Remote Signing and Sealing Service, along with a description of the scenarios in which this service will be utilised.

Pilot	Use case scenario
e-Mandates	Signature generation for e-Mandates: e-Mandates Signature on Debtor's bank - Using the Debtor's Authentication and Signature Token (User-centric Signature Generation with local smart card at User (R1.2.d) – described in D3.6-Design Documentation – Remote Signing and Sealing);

Table 6 - Remote Signing and Sealing Service: links to pilots

4.2.3.6 Alignment planning

Overall, the alignment between the FutureTrust services and pilots will follow a similar approach, going through the following steps:

- Definition of the services interfaces (Work Package 3);
- Implementation of mock external interfaces for initial integration testing;
- Communication of test data by the pilot projects owners to the services owners;
- Integration testing;
- Interfaces refinement, if required.

This approach is illustrated in Figure 7.

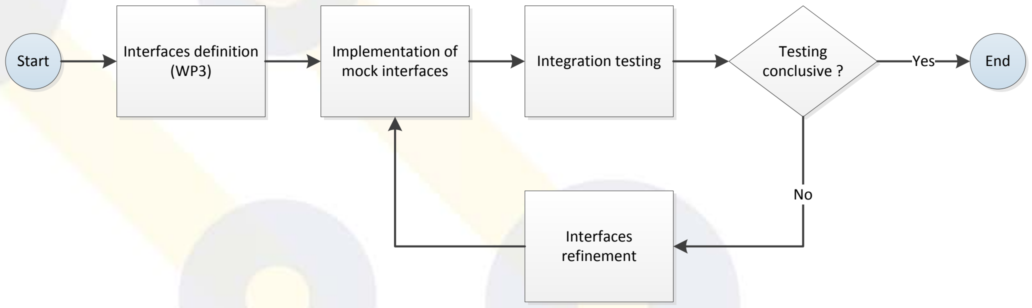


Figure 7 - Pilots and services alignment approach

An indicative alignment planning for the Comprehensive Validation Service is provided in Table 7.

Task	Foreseen start date	Foreseen end date
Interfaces definition	06/2017	30/04/2017
Implementation of mock interfaces	15/05/2017	18/05/2017
Communication of test data	N/A	01/09/2017

Task	Foreseen start date	Foreseen end date
Integration testing	Upon receipt of initial test data	20/04/2018

Table 7 - Comprehensive Validation Service: Pilots alignment planning

An indicative alignment planning for the Scalable Preservation Service is provided in Table 8.

Task	Foreseen start date	Foreseen end date
Interfaces definition	06/2017	30/04/2017
Implementation of mock interfaces	15/05/2017	20/05/2017
Communication of test data	N/A	31/10/2017
Integration testing	Upon receipt of initial test data	23/07/2018

Table 8 - Scalable Preservation Service: Pilots alignment planning

An indicative alignment planning for the Identity Management Service is provided in Table 9.

Task	Foreseen start date	Foreseen end date
Interfaces definition	06/2017	30/04/2017
Implementation of mock interfaces	15/05/2017	20/05/2017
Communication of test data	N/A	31/10/2017
Integration testing	Upon receipt of initial test data	23/07/2018

Table 9 - Identity Management Service: Pilots alignment planning

An indicative alignment planning for the Remote Signing and Sealing Service is provided in Table 10.

Task	Foreseen start date	Foreseen end date
Interfaces definition	06/2017	30/04/2017
Implementation of mock interfaces	15/05/2017	20/05/2017
Communication of test data	N/A	31/10/2017
Integration testing	Upon receipt of initial test data	23/07/2018

Table 10 - Remote Signing and Sealing Service: Pilots alignment planning

Note: there is no alignment planning specified for the Global Trust Service Status List service, as it does not directly interact with any of the pilots.

4.3 eIDAS alignment

The core objective of the FutureTrust project is to support the practical implementation of the eIDAS-regulation (2014/910/EU, 2014) on electronic identification (eID) and trusted services for electronic transactions in the internal market and ease the utilization and proliferation of trustworthy eID and electronic signature technology in Europe and beyond to enable legally significant electronic transactions around the globe.

The following sections provide a definition of high-level requirements identified for the Future Trust services, which were derived from (2014/910/EU, 2014).

4.3.1.1 Electronic Identification requirements

R1. **The Future Trust services SHALL provide the means to support mutual recognition of eID identification means across EU Member States, taking into account the three assurance levels of electronic identification schemes specified under Article 8 of (2014/910/EU, 2014).**

R2. **The Future Trust service(s) enabling eID interoperability as specified in R1 SHALL facilitate the implementation of the principle of privacy by design, and ensure that personal data is processed in accordance with Directive 95/46/EC.**

4.3.1.2 Trust Services requirements

R3. **The Future Trust services SHALL support the inclusion of trust service providers established in a third country in its Trusted Lists scheme.**

4.3.1.3 Electronic Signatures requirements

R4. **The Future Trust services SHALL provide the means to generate advanced electronic signatures complying with the standards allowed under the eIDAS regulation.**

R5. **The Future Trust services SHALL provide the means to validate advanced electronic signatures complying with the standards allowed under the eIDAS regulation.**

R6. **The Future Trust services SHALL provide the means to preserve qualified electronic signatures over prolonged periods.**

4.3.1.4 Electronic Seals requirements

R7. **The Future Trust services SHALL provide the means to generate advanced electronic seals complying with the standards allowed under the eIDAS regulation.**

R8. **The Future Trust services SHALL provide the means to validate advanced electronic seals complying with the standards allowed under the eIDAS regulation.**

R9. **The Future Trust services SHALL provide the means to preserve qualified electronic seals over prolonged periods.**

4.3.1.5 Electronic time-stamps requirements

R10. **The Future Trust services SHALL provide the means to validate electronic time stamps complying with the standards allowed under the eIDAS regulation.**

4.4 Roles and Responsibilities

This section provides a description of the various roles and responsibilities for the implementation phase of the FutureTrust project.

The implementation leader is ARHS Spikeseed (cfr. sections 4, 5, 6, 7, 8, 9, 10 and 12), while the leader for the testing is TUBITAK (cfr.section 11).

4.4.1 Services implementation

Table 11 provides an overview of the various services along with their lead implementer and the relevant contact persons.

Future Trust Service	Sub-task Leader	Contacts
D4.2 – Global Trust List	ARHS Spikeseed	Alexandre Defays (alexandre.defays@arhs-developments.com)
D4.4 – Comprehensive Validation Service	ARHS Spikeseed	Alexandre Defays (alexandre.defays@arhs-developments.com)
D4.5 – Scalable Preservation Service	ECSEC	Detlef Hühnlein (detlef.huehnlein@ecsec.de)
D4.6 – Identity Management Service	G&D	Jens Urmann (jens.urmann@gi-de.com)
D4.7 - Remote Signing and Sealing	ECSEC	Detlef Hühnlein (detlef.huehnlein@ecsec.de)

Table 11 - Future Trust services: contact information

4.4.2 Pilots implementation

Table 12 provides an overview of the various pilot applications along with their lead implementer and the relevant contact persons.

Pilot	Sub-task Leader	Contacts
D4.8 – e-Invoice	BRZ (Austria)	Carl-Markus Piswanger (carl-markus.piswanger@brz.gv.at)
D4.9 – e-Mandates pilot (SEPA direct debit)	MultiCert	Carlos Cardoso (carlos.cardoso@multicert.com)
D4.10 – BVA pilot	BVA (Germany)	Christina Hermanns (Christina.hermanns@bva.bund.de)
D4.11 – eApostille pilot	PSDA (Georgia)	Mikheil Kapanadze (mkapanadze@sda.gov.ge)

Table 12 - Future Trust Pilots: contact information

5. Quality Assurance

The following section describes the overall quality assurance framework that will be set in place for the various implementation activities of the FutureTrust project.

5.1 Quality Process Description

Quality management covers the whole lifecycle of this Work Package and ensures that:

- The quality of deliverables and processes is in line with the requirements of the European Commission;
- Corrective actions are taken in case of deviation;
- The quality of the project is improved on a continuous basis.

The approach that will be followed in terms of quality management is illustrated in Figure 8.

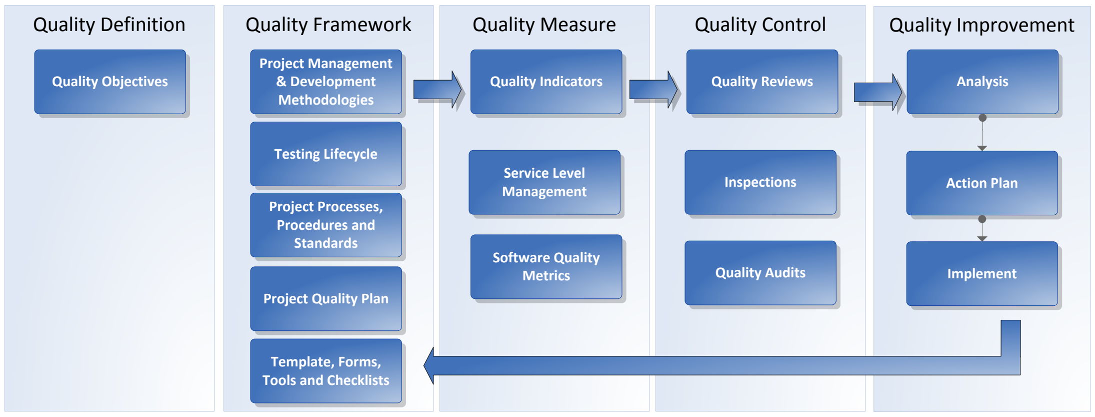


Figure 8 - Quality management approach

5.1.1 Quality Definition

The first step is to precisely define the different Quality Objectives for the project and each component being developed within the project’s scope. This can be done via a Quality Statement that will summarise all key Quality Objectives of the project.

5.1.2 Quality Framework

The Quality Framework covers the different means that will be put in place to ensure that the project meets the Quality Objectives. In software development and maintenance projects, the foundations for Quality of Work rely on the definition and implementation of:

- A **proven and adequate methodology** for Development/Maintenance and for Project Management. The methodology defines the phases, the tasks, the activities and the deliverables;
- An **appropriate Testing Lifecycle** that will ensure that the developed product:
  - Fulfils the **business requirements** and **conforms** to the functional and technical **specifications**;
  - Is **correct, complete**, **robust** and **reliable**;
  - Is **interoperable** with other modules (through integration testing);
  - Does not show any **regression** (non-regression testing);
  - Fulfils **performance** and **security** requirements.
- Detailed and flexible **project processes and procedures** (e.g.: Risk Management, Change Management …);

- **Standards and Templates** for documentation (Deliverables, Meeting Minutes, Progress Reports);
- **Checklists** and their continuous improvement as suggested in CMMI (including checklists for deliveries, etc.);
- **Development Standards** (including Database and Naming Conventions, W3C and XML standards …), **Technical Architecture Standards**, and **User Interface Design Standards** (related to usability, conformity, ergonomics and accessibility).

5.1.3 Quality Measure

To control and improve the quality process, it is necessary to be able to measure it.

- **Quality Indicators.** Quality will be measured through the implementation and follow-up of Key Performance Indicators (KPI), defined with target and limit values, along with the appropriate calculation rules. Quality indicators will mostly be calculated automatically based on a deliverable tracking matrix.
- **Service Level Management** is the process of following up all events related to the quality indicators, calculating and reporting the quality indicators in the Quality Control phase and taking necessary actions to prevent/correct deficiencies during the Quality Improvement phase.
- **Software Quality Metrics.** To ensure robust, maintainable and reusable code we will apply software quality metrics using appropriate tools.

5.1.4 Quality Control

The Quality Control ensures detecting and correcting non-compliances. Removing defects in deliverables as early in the development lifecycle as possible is a key driver for good quality. Quality Control activities cover Quality Reviews, Quality Inspections and Quality audits.

5.1.4.1 Quality Reviews

Quality reviews are an integral part of the implementation workflow which all deliverables should pass before delivery, including:

- **Peer document review** by another project member knowing the details of the task at hand, ensuring the alignment of the document with the requirements and its technical correctness.
- **Quality assurance review** by other project members, ensuring comprehensibility, language and formatting correctness and the compliance with contract and quality requirements.
- **Code review** by an experienced team member or by a Technical Architect in case of key architectural components insuring a high level of code quality in terms of correctness, best practices (including secure coding practices), usage of design patterns, level of documentation and maintainability.

Deliverables being subject to a formal acceptance from the European Commission, must be reviewed by the Task Leader. In specific cases, e.g. confidential design documentation, confidential source code, etc.; ad-hoc reviews will take place, performed by the sub-task or component leader. All deliverables will be subject to the T1/T2/T3 formal review and acceptance procedure as described in section 6.5.

5.1.4.2 Quality Inspections and Quality Audits

Quality inspections will be performed by the Task Leader, except in specific cases (e.g. confidentiality agreements). The main goal is to check the compliance of the project team with the quality objectives and with the project standards and procedures focussing on

- The **results** of a key milestone in the project to check that all **expected results** are delivered, that the documentation is **complete**, etc.;
- A specific **process** (Factory Acceptance Testing, Coding and Unit Testing, Issues Management) to check the **adherence to the defined standards** and procedures; and
- The **quality of software coding** and **compliance** with configuration management principles.

Such quality inspections and audits will be adapted in specific cases (e.g. confidential source code, designs etc.).

The key for successful quality inspections is the development of **specific checklists** (e.g. SVN delivery process, document review process, source code quality, etc.) associated with the major milestones or with the main processes to verify.

5.1.5 Quality Improvement

Quality management goes further than checking the compliance with the project standards, procedures and the quality requirements. It is essential to put in place a process that identifies and implements the opportunities for improvement.

Suggestions for quality improvements will be raised from different sources: the quality control process, suggestions from project stakeholders, preventive maintenance or suggestions from the European Commission. Meetings will be organised regularly and will gather the Task Leader and key task stakeholders.

The purpose is to review the quality indicators and suggestions for improvements, agree on new actions to correct the deficiencies or to improve the services and follow-up implementation of the action plan.

6. Standards and Methods

This approach has been built based on recognised standards and methodologies such as ISO 9001, SCRUM, RUP, CMMI, PMBOK, ISO 20000 (ITIL), ISO 27001 and PRINCE2 and is tuned to the specific needs of this project.

6.1 Software Development

The methodologies applied throughout the development cycle of the various components vary from one implementer to the other. Table 13 provides an overview of the methodologies that will be used, per involved stakeholder and task.

Stakeholder	Component(s)	Methodologies
ARHS	T4.2 – Global Trust Service Status List	

Stakeholder	Component(s)	Methodologies
	T4.4 – Comprehensive Validation Service	Agile (Scrum), with Continuous Integration
BRZ	D4.9: eInvoice Pilot	Agile with CD/CI
ECSEC	T4.5: Scalable Preservation Service	Iterative with Continuous Integration
	T4.6: Identity Management Service	
	T4.7: Remote Signing and Sealing	
G&D	T4.6: Identity Management Service	Agile (Scrum), with Continuous Integration
MULTICERT	D4.8: SEPA e-Mandates Demonstrator	Agile with Continuous Integration
PSDA	T4.11: e-Apostille pilot	Agile
RUB	T4.3: Comprehensive Validation Service	Prototyping
Trustable	T4.4 – Comprehensive Validation Service	CDCI
	T4.5 – Scalable Preservation Service	
	T4.7 – Remote Signing and Sealing Service	
TUBITAK	T4.2: Global Trust Service Status List	Agile
	T4.5: Identity Management Service	
	T4.6: Remote Signing and Sealing	

Table 13 - Software development methodologies

6.2 Service Management

The Service Management approach is based on the ITIL v3 best practices and includes:

- **Helpdesk.** Single Point of Contact (SPOC) for incident reception, which is the task leader of the service to which the incident is related.
- **Incident Management.** Recording, classification, resolving of incidents.
- **Problem Management.** In-depth problem analysis for long-term resolution.
- **Change Management.** Categorisation, analysis and planning of changes.
- **Release and Deployment Management.** Planning, management, developing, testing and implementation of changes in releases.
- **Service Asset and Configuration Management.** Controlled management of configuration items.

6.3 Risk Management

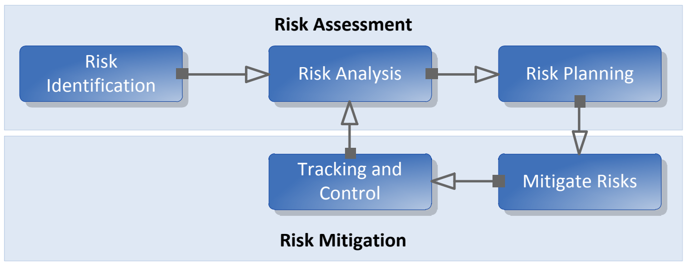


Figure 9 - Risk Management Process

The WP Leader is responsible for the risk management process. He appoints a risk owner for the follow-up of an action plan and the successful outcome of the mitigating actions. All risks (including description, status, probability, impact, owner and mitigation/contingency action plan) are logged in the Risk Management Log that will be reviewed each month by the Project Leader.

6.4 Delivery Management

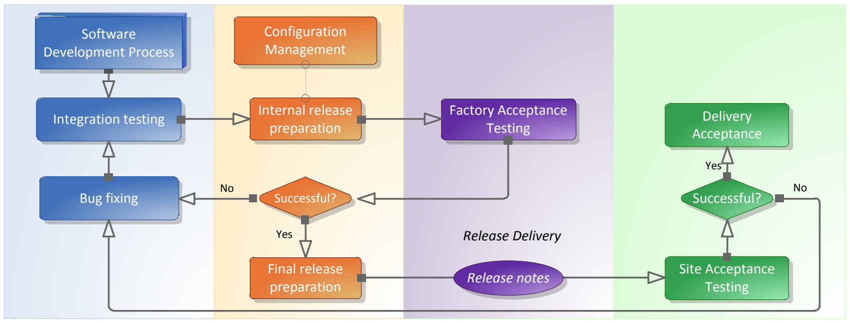


Figure 10 - Delivery Management Process

The Delivery Management Process related to software deliverables, illustrated in Figure 10, includes:

- **Integration Testing.** Prior to preparing a release for delivery, integration testing is performed. This is also performed in parallel to the development activities as part of a **continuous integration** procedure.
- **Release Preparation.** The release will be assembled from all its artefacts stored in Configuration Management.
- **Acceptance Testing.** Testing the complete system against its technical and functional requirements.

6.5 Documentation Management

The Documentation Management Process is based on:

- A **strategy** identifying the documentation to be produced during the lifecycle of the software product or service;
- **Full traceability** of requirements (functional and technical) from the contract requirements to the specifications, implementation and test cases;
- The **standards** to be applied for the development of the software documentation.

The so-called **T1/T2/T3 review cycle** will be applied on documentation deliverables as illustrated in Figure 11.

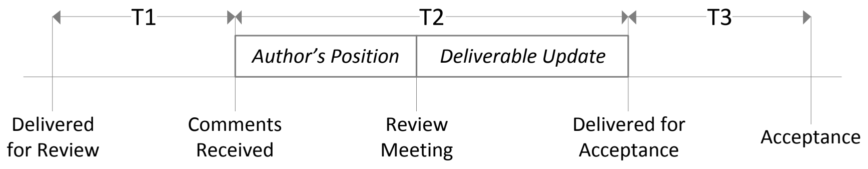


Figure 11 - T1/T2/T3 review cycle

See section 10 for details regarding documentation production and quality assurance.

6.6 Communication Management

The key principles of the Communication Management Process are the clear documentation of all stakeholders, communication paths and communication means.

7. Issue Handling Approach

The issue handling approach is based on the ITIL v3 processes Incident Management, Problem Management, Change Management, Service Asset and Configuration Management, and Release and Deployment Management as globally depicted in the following diagram. These processes are supported by the application of an ITIL based Knowledge Management.

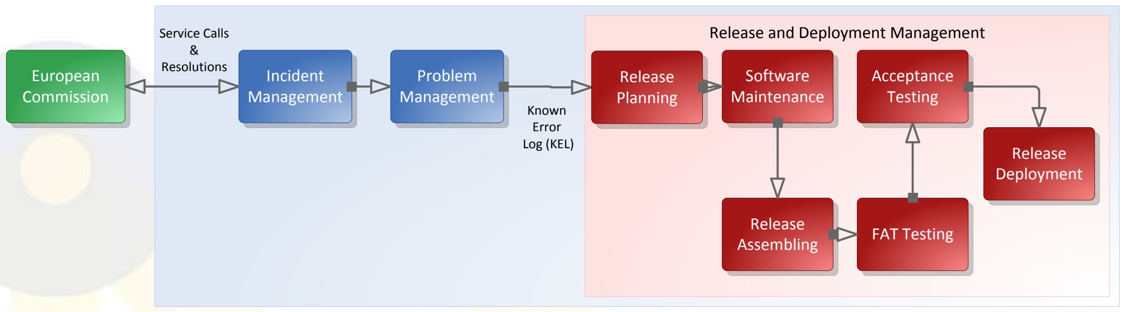


Figure 12 - Issue Handling Approach

This approach will ensure:

- **Stable operation** of the various services and pilots;
- **Quick reactions** in case of reported incidents;
- **Thorough analysis** of root causes of reported incidents;
- **Preventive maintenance** through the identification of possible issues before they impact users;
- **Full traceability** from issues to requirements to source code changes to tests to the final release.

In practical terms, technical issues will be processed along the following steps:

- **Collect tracing information** (if available) on the client and server sides produced when the problem arises.

- **Set-up the local test environment** corresponding to the application version and execution conditions reported with the problem.
- Try to **reproduce the same problem** in this controlled test environment and track the phenomenon with appropriate tools (proxy, network sniffer, database analyser, debugger …) and actions (manual steps to reproduce, traffic simulator, connection interruption …).

8. Handling of Maintenance Requests

The process of handling maintenance requests covers the different possible requests that could be received during the execution of this project, including:

- **Corrective Maintenance** for high, medium and low priority issues;
- **Preventive Maintenance** performed periodically;
- **Remote Maintenance** services provided from our offices;
- **Evolutionary Maintenance** for major or minor evolutions.

Corrective maintenance requests are handled through the issue handling processes described in section 7. Preventive maintenance will be performed periodically as needed by the development teams, e.g. for platform upgrades.

9. Software Development and Maintenance Tools

This chapter provides an insight on the various software development and code maintenance tools that are used by the stakeholders involved in the development tasks. A specific questionnaire was provided to gather the specific tools that are used by each stakeholder. The answers to this questionnaire have been summarized and listed in Table 14.

Note: "NA" indicates that the stakeholder does not currently use the particular technology, and is open to partner recommendations.

Stakeholder	Component(s)	Continuous Integration	Build Engine	Code Hosting	Code Review	Runtime Environment	Software Repository	Code Quality Analysis	Deployment Container
ARHS	T4.2 (gTSL) T4.3 (ValS)	Jenkins	Maven	Private GitLab	GitLab	JVM	Nexus	SonarQube findBugs CheckStyle	Docker
BRZ	eInvoice Pilot	Jenkins	Maven	gitHub	Crucible	JVM	Archiva	findBugs	vmWare
ECSEC	T4.4 (PresS) T4.5 (idMS) T4.6 (mSignS)	Jenkins	Maven	Private GitLab	GitLab	JVM TypeScript	Nexus	findBugs CheckStyle	KVM
G&D	T4.5 (idMS)	Jenkins	Maven, Gradle	BitBucket	BitBucket	JVM Native Support C/C++	Nexus	SonarQube	NA
Multicert	SEPA eMandate Pilot	Jenkins	Maven	Private Git	Crucible	JVM	Nexus	SonarQube	NA
PSDA	eApostille Pilot	Jenkins	Maven	Git, Private SVN	NA	JVM	Artifactory	NA	vmWare
RUB	T4.3 (ValS)	NA	Maven	gitHub	NA	JVM Javascript	NA	findBugs CheckStyle	NA

Stakeholder	Component(s)	Continuous Integration	Build Engine	Code Hosting	Code Review	Runtime Environment	Software Repository	Code Quality Analysis	Deployment Container
								PMD	
TRUSTABLE	T4.3 (ValS) T4.4 (PresS) T4.5 (idMS) T4.6 (mSignS)	Jenkins	Maven	gitHub	NA	JVM	Artifactory	SonarQube	VirtualBox
TUBITAK	T4.2 (gTSL) T4.5 (idMS) T4.6 (mSignS)	Jenkins	Maven, Gradle	gitHub, Private Gitolite	NA	JVM Javascript	Nexus	SonarQube	VirtualBox

Table 14 - Overview of software development and maintenance tools

9.1 Coding Quality Verification and Metrics

The quality of the delivered software depends on a sound design and the application of high development and source code quality standards.

Software components will be designed in order to be **easily usable** (*quality metrics* based on **learnability**, **efficiency**, **memorability**, **error handling**, **user satisfaction**), **configurable**, **supportable**, **extensible** and **portable** (*quality metrics* based on **supported platforms**, **standards**, **accessible interfaces** vs. hidden functionality).

10. Documentation Production Procedure

The documentation involved in an IT/IS implementation system covers a wide range including user requirements, detailed architectural designs, scope documents, functional specifications, feasibility studies, etc. Different documents should be produced at the different stages of a project.

Find hereafter a list of documents which should be produced during the different phases of the project:

Inception

- Implementation and Test Plan.

Construction

- Installation Instructions
- User Manual;
- Operation Manual;
- Release Notes.

Elaboration

- Functional Analysis (Business Modelling) and Software Architecture;

Transition

- FAT report.

Some other documents (e.g. the Progress Report) are not dependent on a specific phase and should be produced during the whole project duration. During the entire process, the Task Leader should ensure that the documentation meets the agreed requirements.

Please note that some documents of the elaboration phase as well as source code may be company confidential and not available to other partners. This confidentiality is required e.g. in order to exploit the FutureTrust components in products and services which require a security certification.

Our Documentation Production Procedure is illustrated in Figure 14:

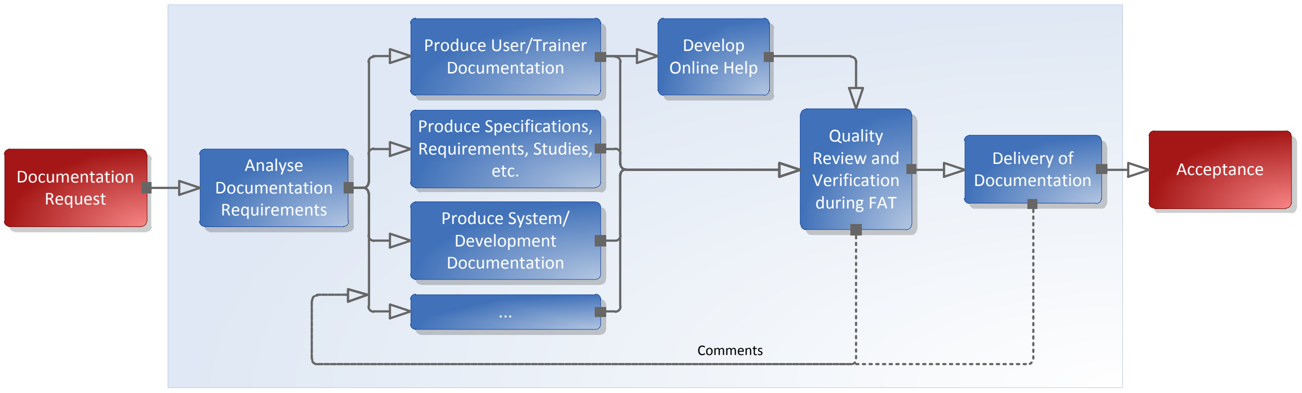

Documentation Request
Analyse Documentation Requirements
Produce User/Trainer Documentation
Produce Specifications, Requirements, Studies, etc.
Produce System/ Development Documentation
...
Develop Online Help
Quality Review and Verification during FAT
Delivery of Documentation
Acceptance
Comments

Figure 13 - Documentation Production Procedure

10.1 Documentation Production

First of all, the author needs to create a draft version of the document to be produced. Documentation is written using Microsoft Word, or generated by automated tools when applicable. The author of a document has full knowledge of the area to be described. The author needs to draw special attention on the

- **Completeness** and **Comprehension** of the content;
- **Clear** and plain English and/or French **language**;
- **Correct versioning** of the document according to the requirements and application of the naming conventions;
- The **update of all properties** of the document as the release dates;
- **Documentation checklists** with comments or requests from previous documents;
- Correct use of the **templates**, if any.

10.2 Quality Review

Once produced, the draft version is reviewed by a qualified team member. The quality reviewer needs to:

- **Review and update** the content for adequacy;
- **Identify the changes** and current document revision status;
- Ensure the documents remain **legible and correctly identifiable**;
- Verify that the document follows the applicable **naming convention and template**, if any;
- Update and verify documentation **checklists**.

After the review, the document is updated by the author. Once passed through the internal quality review, the document is sent for review to the customer. Afterwards, the review cycle (defined in section 6.5 Documentation Management) takes place. Comments can be communicated using the MS Word comments or track changes functionality.

11. Test Plan

Defects may arise in software because of incorrect design, coding, setup, usage and any action that a human is involved in. Testing is a crucial step of a software development life cycle (SDLC), starting from the unit testing through system testing to reveal defects and errors that are made during implementation. Software Development without a proper and adequate testing leads to low software quality, high maintenance cost, unreliable and inaccurate results and eventually client dissatisfaction and loss of reputation.

A common SDLC includes unit testing for every fraction of code, integration testing on a regular basis and system testing. It is important to find and solve problems in early stages of a development cycle because handling these problems in the later stages can be very expensive. In other words, as in bottom-up approach and starting from the unit testing, each level of testing reduces uncertainty in its level and makes testing in next higher level easier. Therefore, we provide the unit test plan and integration test plan sections just to provide some suggestions and encourage the Development Team (DevTeam) to apply such tests in Future Trust component implementation works.

The present test plan section describes the scope, methodology and required resources of the testing activities among the FutureTrust project. The test plan of FutureTrust project is constructed based on (ISO/IEC/IEEE 29119, 2013) by tailoring to the project, while the test documentation will be in line (ISO/IEC/IEEE 29119, 2013), Part 3. The Master Test Plan provides the overall test planning and unifies the other level-specific test plans, namely Unit Test Plan, Integration Test Plan and System Test Plan.

11.1 Roles and Responsibilities

Action	DesignTeam	DevTeam	TestTeam
Analysis and Design of System Modules	X		
Analysis and Design of Software Modules	X	X	
Preparing Master Test Plan			X
Preparing Unit Testing Plan		X	X
Preparing Integration Testing Plan		X	X
Preparing System Testing Plan			X
Applying Unit Tests		X	
Applying Integration Tests		X	
Preparing Components for System Tests		X	
Applying System Tests			X

Table 15 - Roles and Responsibilities for SDLC

Design Team (DesignTeam): WP3 is the analysis and design package, of which members form DesignTeam, are responsible for the following items;

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700542

- Generating system requirements out of related standards and pilot applications,
- Designing the system according to system requirements
- Generating software requirements from the result of system analysis and design
- Designing software modules
- Creating Test Assertions from system's functional requirements for system testing

Development Team (DevTeam): WP4 is the implementation and testing package, of which members form DevTeam, are responsible for the following items;

- Implementing designed software modules
- Determining the technologies that will be used during the implementation
- Producing the planned implementation documents
- Performing Unit Tests (Preferably)
- Performing Integration Tests (Preferably)

Test Team (TestTeam): WP4 is the implementation and testing package. Some of the members embody TestTeam, which is responsible for the following items;

- Determining the technologies that will be used during the system testing
- Implementing means for system testing
- Generating Master Test Plan Document that includes Unit Testing Plan, Integration Testing Plan and System Testing Plan.
- Applying system tests

It is important to note that, unit test plan and integration test plan aim just to give some suggestions about writing proper unit and integration tests for reaching an adequate software quality and declare the DevTeam's preferred toolset.

11.2 Test Deliverables

In M36, TestTeam is planned to provide the deliverable "D4.12 Conformance and Interoperability Testing Results" that will contain the adopted Minder architecture to apply system tests for FutureTrust components and results of system testing activities on Minder.

11.3 Unit Testing Plan

Unit testing aims to verify the behaviour of a code fraction independently from other parts for a given set of inputs. Continuous integration advises unit testing at every commit to prevent changes from breaking the functionality. A unit test can be "state-based" or "interaction based" and it has mainly three phases called as Arrange, Act, Assert (AAA);

i. Arrange: Initializing the related pieces of an application
ii. Act: Triggering some events according to the test case scenario
iii. Assert: Observing the results

Coding unit tests after an implementation is quite widespread among software developers. However, unit testing can fit its purpose when the code fraction under test is coded in a testable way. This test plan provides some suggestions about writing testable code and testing rules. It aims to encourage the DevTeam to apply unit testing in order to increase overall quality of the components. The developer (who has the full knowledge of the components) is responsible for the specification and execution of the unit tests. As much as possible, he will define unit test cases that are independent one from another, and may use mock objects in order to assist to the testing of an isolated module.

Unit testing can either be executed manually or automatically (by using tools such as JUnit for Java applications or HTTPUnit for web applications).

11.4 Items To Be Tested

Each relatively small piece of code needs to be tested independently from other parts of the software at each commit. As a common practice, a developer is responsible for writing unit tests of what she/he developed. An Ideal unit test should be easy to write, readable, reliable, fast and truly unit.

Unit tests are vital for ensuring that every small piece of code works correctly. Therefore, it is important for a unit test writer to cover all cases and not to confuse unit testing with integration testing. Unit testing does not use real collaborators for a method or a class, instead, uses mocking. So, using real database connections or a real collaborator's object are not true approaches in unit testing. The integration with other real resources is actually subject to integration testing.

11.5 Test Approach

11.5.1 Testing Framework

Including open-source ones, there are many third-party unit testing frameworks for various languages. Currently, programming languages and technologies are not determined for Future Trust modules' implementations.

11.5.2 How to Write Testable Code

The first step of writing a unit test actually starts by developing testable code fractions. It is important for a developer considering how to test a code when writing or reviewing it. If testing a code fragment seems hard, developer should take this as a warning. The items that should be avoided during an implementation to be able to write testable codes are given in Table 16. The listed items are constituted from the study (Jonathan Wolter, 2008), that includes detailed explanations and examples.

Flaw	Flaw Result	What to avoid
Overloaded Constructors	1. Inflexible, 2. Causes coupling, 3. Violates Single Responsibility Principle,	1. "new" keyword inside a constructor 2. "new" keyword at field declaration 3. Static method calls in a constructor 4. Static method calls at field declaration 5. Coding some logics in constructor other than field assignment

Flaw	Flaw Result	What to avoid
	4. May cause unchecked design 5. Hard to mock collaborators that called inside the constructor and slower test runs	6. Conditional or looping logic inside a constructor 7. Creating collaborators inside a constructor (Use builder, factory or injection instead). 8. Adding an initialization block inside a constructor
Method Call Chain Over Collaborators	1. Violates Law of Demeter	1. Passing objects that used only for accessing to other objects. 2. Calling methods in a chain using „.“
Global State and Singleton	1.Static methods/fields that cannot be mocked	1. Using singletons 2. Using static fields and methods 3. Using static initialization methods 4. Using service locators 5. Using registries
Overloaded Classes	1.Hard to understand, debug, test 2. Not extensible 3. High coupling	1. Classes that have many responsibilities 2. Excessive code lines 3. Many collaborators

Table 16 - What to Avoid For Creating Testable Code

11.6 Item Pass/Fail Criteria

The observed behaviour of the code fragment must be consistent with the expected result.

11.7 Integration Testing Plan

Having a module work fine individually does not guarantee that it will work correctly when integrated. Bugs that arise from the integration within a module cannot be determined by unit testing. In order to prevent such bugs, integration testing should take place in a regular basis. Integration testing provides means for testing when the two or more modules are combined and aims to decrease the time required for other types of testing that following integration testing. Integration tests are similar to unit tests, except they do not use mocking for other components but use the real ones. For example, while unit testing does not use a real database but mocks it, integration testing should use a real database. Since integration testing may require an initial setup for some components, complexity increases and running tests takes more time than unit tests. The same tools as unit tests are usually used for integration tests. Note that, integration testing within a module is in the responsibility of its DevTeam members.

11.8 System Testing Plan

System testing is black-box testing process that is executed after all modules are developed and integrated. It verifies whether the software as a whole is developed according to the functional and non-functional system requirements of the project. In the FutureTrust project, system testing includes the Conformance and Interoperability Testing, the scope of which is specified with the following definitions:

I. Conformance Testing verifies that a Future Trust component complies with the project's functional system requirements.
II. Interoperability Testing verifies the ability of two Future Trust components' to interact with each other correctly.

While Unit Testing and Integration Testing are in the responsibility of the WP4 DevTeam members, the TestTeam is responsible for system testing. Since system tests can be difficult to write and maintain due to their very high complexity, all FutureTrust components to be tested are expected to be stable and have robust releases to apply system tests. The releases are taken as versions and each test cycle is traced according to these releases.

In the Future Trust project, System Tests will be performed on Minder, which is a generic and modular web-based TestBed. It provides a set of software architecture, platform, tools, supplementary testing assets and methodologies to facilitate and standardize the testing procedures in order to make the process reusable and sustainable. The general architecture of the test environment is given in Figure 15. For more details, it is planned to release "Minder Test Developer API Guide", "Minder Test Designer Guide**"** for referring.

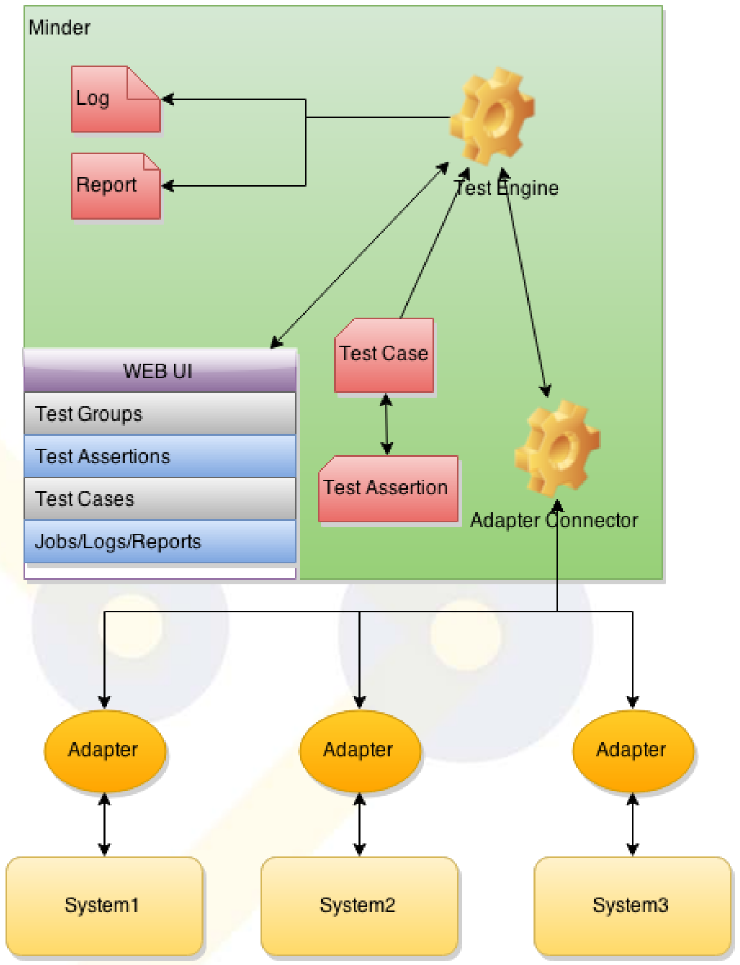


Figure 14 - Minder Applied Architecture

When a system is subject to system testing, the test assertion version and SUT/Asset version are taken as unique test iteration labels and testing will be performed in a test cycle. In each cycle, test results will be generated and delivered. If any major defects/bugs that affect the operation of the system, are found; they are reported to its developers and considered as *stoppers*, and no further system testing will be performed on that product until those stopper defects/bugs are fixed. In the next cycle, it is assumed that the defect/bug is fixed and the system test is repeated.

Test methodology is composed of the following activities:

- **Creation of Test Assertions**

  OASIS Test Assertions Specification (Durand, 2012) defines test assertions as follows:

  "A test assertion is a testable or measurable expression for evaluating the adherence of an implementation (or part of it) to one or more normative statements in a specification".

  Test assertions (TAs) state the testable logics of a system under test. It is planned to create OASIS Test Assertion compatible Future Trust test assertions, which are generated from Future Trust System Functional Requirements.

- **Creation of Adapters**

  DevTeam must provide an adapter implementation for each component in order to be plugged into the Minder testing architecture. TestTeam provides all the necessary documentation to guide developers through the integration of implementations to the system testing environment. Moreover, the team may provide technical help for adapter development in terms of Minder connection.

- **Creation of Test Cases**

  After creating TAs and adapters, TestTeam as test designers can write test cases on Minder. Test cases represent either the whole or a part of a test assertion predicate interpreted in the MTDL that can be run on a concrete target (SUT). TAs that are related to each other are grouped under a *Test Group* (the TestAssertionSet in OASIS TAML (Stephen D. Green, 2011).

- **Execution of the Tests**

  After the creation of test cases and mapping to actual SUTs, test cases are executed from the Minder management GUI. After each execution Minder creates logs and reports for that specific execution

- **Reporting**

  The test execution reports are published on the necessary repositories and provided to the consideration of the observing entities (DevTeam, Future Trust Management, pilots…).

11.8.1 Operational Roles

1. **Test Designer**: is responsible for creating test scenarios according to the test assertions, which are supposed to be obtained from WP3 or the providers of the inherited specification. After creating test scenarios, TS can also run the test cases. TS can communicate with Minder Server via a web interface.
2. **Test Developer:** is responsible in creating the bridge between the SUT and the testing environment. She/He knows all the communication interfaces of the SUT. A TD is responsible to implement the adapter interface provided by Minder. If the SUT uses a standard communication interface (i.e. Rest Calls etc.), the TD might also reuse an existing minder adapter from the Minder Repository.
3. **Solution Providers:** A Solution Provider can be a WP4 participant, which provides implementations for Future Trust components. A solution provider provides technical support for the implementation of the SUT-Minder adapter (or reuse of an existing one) as Test Developer.
4. **Component Experts**: are member of WP3 and responsible to give support for the generation of the test assertions according to the project's system functional requirements.

11.8.2 Test Approach

Future Trust system testing activities will be performed via Minder TestBed. Throughout the whole system testing, the basic assumption is that, the products that are to be tested have already completed the unit/integration tests and the stable releases have been published. This is because due to the fact that system testing is not a development phase testing. Therefore, versions such as beta, snapshot, alpha, pre-release and any other statement that implies a non-release version is under the responsibility of the DevTeam.

11.8.3 Item Pass/Fail Criteria

Test criteria include pass/fail, suspension and resumption criteria. The criteria are taken from OASIS TAM

The completion criterion for System Integration iteration is the execution of the all test cases generated from test assertions in a test cycle. Test cases should cover all the test assertions and they should be in written in a sufficient level of detail.

Each test case execution can have two possible results: Pass **(***success***)** or Fail. Any additional warning will be logged in the report.

The overall PASS criterion for a test cycle is the *success* of all the *mandatory* test cases. The mandatory states of the test cases are derived from the *prescription level* field of the test assertions. There are three prescription levels defined in the OASIS TAM: i) mandatory, ii) permitted and iii) preferred. The *permitted* and *preferred* test assertions are considered as optional, and the FAIL state of their test cases cause the test cycle to PASS with warnings.

11.8.4 Outputs of System Testing

Outputs are the artefacts that are generated and given to the stakeholders participating in tests. The outputs can be listed as follows for system testing:

1. ***Test Execution/Summary Report:*** After test cases are executed on Minder, the results of the test cases are documented in a report. On the report, the details of the test assertion, test run status, the total result of the test, etc. information is included.
2. ***Test Results:*** Error and Execution Logs that are generated after executing test cases on Minder
3. ***Test Cases:*** Test cases that are derived from test assertions to be executed on Minder.

11.8.5 Traceability

The system testing activities produce results that need to be accessible to all the observers. In order to achieve a long-term sustainability of the test deliverables, an identification pattern for the test results is proposed. On the other hand, there is no planned traceability scheme for unit and integration test

Each test cycle takes test assertion details and SUT information (name, version) as an input. In order to achieve traceability for testing events, each testing activity must be labelled with the following identifiers:

- Test assertion set version
- Test assertion ID
- SUT/asset name and version
- Short result (success/fail)
- Report Reference
- Date

12. Infrastructure

The present section gives a brief overview of the **infrastructure that will be used during the execution of the tasks**. This description is limited only to the technical infrastructure relevant to the context at hand without going into irrelevant explanations (e.g. network, office facilities, locations, etc.).

Principally, three types of environments will be setup by the stakeholders:

- **Development environment** used by the development teams for daily development activities;
- **Continuous integration environment** for the continuous builds and tests;
- **Test and demonstration environment** used for testing, reproduction and demonstration purposes.

The test and demonstration environments will be exposed on the Internet, allowing all interested parties to interact with them.

As shown in Table 14, the technological choices vary from one implementer to the other. However, most stakeholders rely on JVM-based runtime environments, while Jenkins is the main Continuous Integration tool used by the stakeholders. Furthermore, virtualization software and application containers are also emerging choices among the stakeholders.

13. References

2014/910/EU. (2014). Regulation (EU) No 910/2014 of the European Parliament and of the council on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2014.257.01.0073.01.ENG.

Durand, J. (2012, July 27). *OASIS:Conformance, Interoperability and Portability Testing: Proposed Procedures and Practices*. Retrieved December 8, 2016, from https://wiki.oasis-open.org/tab/TestingPolicy

ISO/IEC/IEEE 29119. (2013). Software and Systems Engineering -- Software Testing.

Jonathan Wolter, R. R. (2008, November). *Guide: Writing Testable Code*. (Google) Retrieved December 2016, from http://misko.hevery.com/attachments/Guide-Writing%20Testable%20Code.pdf

Stephen D. Green, J. D. (2011, November 30). *OASIS:Test Assertions Part 2 - Test Assertion Markup Language Version 1.0*. Retrieved April 7, 2017, from http://docs.oasis-open.org/tag/taml/v1.0/testassertionmarkuplanguage-1.0.html