success

D3.10v1.0

Integration and Validation Plan. Test and certification specifications

The research leading to these results has received funding from the European Union's Horizon 2020 Research and Innovation Programme, under Grant Agreement no 700416.

Project Name	success
Contractual Delivery Date:	31.01.2017
Actual Delivery Date:	**31.01.2017**
Contributors:	ENG, SYN, P3C, RWTH, ISMB
Workpackage:	WP3 – Securing Smart Devices
Estimated person months:	3.75
Security:	PU
Nature:	R
Version:	1.0
Total number of pages:	18

Abstract:

This deliverable presents the initial results from Task T3.5 and T3.6 and pertains to the integration of components of WP3, embodying the interfacing between the DSO Security Monitoring Centre, NORM and the Distribution Grid level. Moreover, this report outlines the necessary processes and the associated methodology towards integrating and testing, at functional level, between WP3 components within the success Security Monitoring Solution.

Keyword list:

Security, communication, Utility, Architecture, Threat, Countermeasure, Integration, Validation, Testing

Executive Summary

The purpose of this deliverable is to present an initial plan as to the testing of the SUCCESS Security Monitoring Solution components produced in WP3, i.e. the DSOSMC and the NORM as well as to provide a plan for the integration of both the component belonging to WP3.

The focus of this deliverable is upon WP3, “Securing Smart Devices”, and, specifically, on Task T3.5 entitled “Information Security Management Integration and Testing” and on Task 3.6 “Certification Feature Catalogue, Feature Specifications and Test Plans”. This deliverable presents the structure of the individual tests (chapter 3) planned to be performed for the WP3 components (NORM and DSOSMC) and also an integration plan in order to verify that both the components work and interact as expected (chapter4). Another important aspect covered is the certification process of these components which is closely related to the tests performed for the integration plan. The level of details provided for testing and integration will be updated as the components become available, the specifications are finally concluded and the integration process starts. For this reason, this deliverable will be followed by a new (updated) version in which these details will be provided along with the test results and corrective measures taken, where relevant.

Authors

Partner	Name	e-mail
P3C		
	Paschalidis, Panagiotis	Panagiotis.Paschalidis@p3-group.com
ENGINEERING (ENG)		
	Antonello Corsi	antonello.corsi@eng.it
	Giampaolo Fiorentino	giampaolo.fiorentino@eng.it
SYNELIXIS (SYN)		
	Sotiris Karachontzitis	karachontzitis@synelixis.com
	Artemis Voulkidis	voulkidis@synelixis.com
Rheinisch-Westfälische Technische Hochschule Aachen(RWTH)		
	Padraic McKeever	PMcKeever@eonerc.rwth-aachen.de
ISMB		
	Mikhail Simonov	simonov@ismb.it

Table of Contents

1. Introduction

The entire integrated success Security Monitoring Solution includes a Pan-European Security Monitoring Centre (ESMC), consisting of a central instance, namely the Security Monitoring and Information System (E-SMIS), and several further instances distributed across Europe, namely the DE-SMIS instances. Both, E-SMIS and DE-SMIS, interwork to detect patterns at pan-European level related to cyber-attacks (and physical ones, at a more limited scale) and to alert DSOs or TSOs about attacks. In synergy with this solution, success is also developing a Decision Support System for DSOs entitled “DSO Security Monitoring Centre” (DSOSMC) which, based on data gathered from the distribution grid at Neighbourhood area Network (NaN) level, identifies attacks, performs risk assessment and, finally, suggests relevant, proper countermeasures. In the context of the success infrastructure, data are being provided from NORM devices (the Smart Meter Gateway being developed by success). The DSOSMCs send data towards the local DE-SMIS instances with information on identified threats, applied countermeasures and grid status. In turn, the DE-SMIS instances process this information by interacting with E-SMIS and by resorting to further data sources. Another component that can also implement the countermeasures is the Breakout Gateway, which is a new mobile communications node being developed in success, which allows mobile core network functionality. Further details on the E-SMIS, DE-SMIS and the Breakout Gateway may be found in [2].

With the purpose of testing and integration of the main WP3 outcomes we will provide a hierarchical approach based first on testing single components and then we will proceed with the integration step that involves the two components of the system working together. This approach will pave the way also to the certification process of the Success Solution to be held in [2].

In general, the target of any testing procedure is to identify software errors and bugs, hardware deficiencies and unexpected behaviour and to provide corrective measures in a timely manner. It should be stressed that the testing procedures, regarding only components developed in WP3 will described in this deliverable while the simulation with real users, will be held in the trial sites of the success Security Monitoring Solution, as developed in D4.4 and will be the first outcome of WP5 and will be elaborated therein.

This deliverable together with D3.11 [3] and D3.12 [4] will constitute the main outcome of T3.5 – *Information Security Management Integration and Testing* and of T3.6 - *Certification Feature Catalogue, Feature Specifications and Test Plans.* The purpose of T3.5 is to provide a validation and integration plan for the SUCCESS security monitoring solution, whereas the purpose of T3.6 is to define the specifications for the certification of the components developed in WP3.
In this deliverable, three main objectives are pursued:

- To provide a preliminary plan to be adopted both in D3.11 [3] and D3.12 [4] and also offer a trace for the testing, and integration of success security Monitoring Solution that will be addresses in the deliverables D4.7 [1], D4.8 [5] and D4.9 [6]
- To furnish a preliminary version of the individual WP3 component testing.

This group of six deliverables that correspond to WP3 and WP4 outcomes will form the starting point from which the WP5 will implement and manage the deployment of the success Security Monitoring Solution in the real success trial sites.

This deliverable offers an initial integration and testing framework given that the components to be tested are not yet developed. For this reason, it should be highlighted that the level of details provided for testing will be updated as the components become available and the integration process starts.

The rest of the document is structured as follows: Chapter 2 introduces a brief state of the art information related to integration and testing. Chapter 3 provides the initial plan for the testing phase of individual components of NORM [7] and DSOSMC [8]. Moreover, it describes a set of case template that can be used to describe the tests to be performed on the success Security

Monitoring Solution components, before and during the integration phase. The templates contain information that will be populated when the integrated modules are delivered. Finally, Chapter 4 provides a preliminary approach towards the definition of the integration plan for NORM and DSOSMC. Here we use the same approach as in WP4, which is described in D4.7 [1].

2. State of the Art integration and validation strategies

In this chapter, a state-of-the-art coverage of modern approaches towards effective integration and testing procedures is presented, followed by a description of the relevant strategy adopted by the success consortium members of WP3 (ENG, SYN, ISMB, CRE, TMW,P3C).

2.1 Component testing

Component testing is defined as a method used to test the functionality of an individual component. In the context of the success, component testing corresponds to testing the individual components of the success Security Monitoring Solution to determine if their implementation matches the related requirements [9]. This is the first stage of software testing process. Since it is an early phase, it could help to find out bugs and logical errors so they could be fixed and are not propagated to other stages of the testing. Units are the smallest parts of the system. In this context, a unit is a component of success. Tests for this units could be developed by a programmer or by a white-box tester and sometimes unit tests are developed before system itself which is called Test-driven development [10]. All the tested units have to pass the tests so the developers could be ensured that the system meets specified requirements. In an ideal case, all the methods and functionalities should be tested by the unit tests. This is impossible in the real test development because of the complexity of the tested system. However, we can still achieve a high level of code coverage.

2.1.1 Testing Methods

Depending on the information available to the tester, software testing methods are divided into:

- white-box testing.
- black-box testing.
- grey-box testing.

2.1.1.1 White-box testing

White-box [11] testing is used for the testing a system at the level of the source code. It is a detailed inspection of an internal logic which verifies the correct implementation of internal units, structures and relationships between them. A tester needs to have deep knowledge about the system being tested including its internal structure, design and implementation. An internal perspective of the system, as well as programming skills, are very important and used to create test cases. The basic step of white-box testing is that the tester must choose an input which involves different types of requirements, functional specifications, etc. In the next step, the test cases are built to make sure they thoroughly test the application (i.e. all functionalities expected are under investigation) and that given results are correct and as expected. The last step of white-box testing is a report the generation of a report that contains all the inputs from the preparation phase and the results.

This approach brings the following advantages:

- software can be thoroughly tested due to having the knowledge of the source code, and because it is easier to find out which kind of input could test the application properly;
- the source code can be optimized due to errors and defects determination;

Simultaneously, it also bears the following disadvantages:

- white-box testing adds complexity to the testing procedures, so a programmer with good skills and high-level of knowledge is required, implying increasing costs;
- in large scale and complex systems it is unrealistic to be able to test every single existing condition, hence some conditions and parts of the system might stay untested.

2.1.1.2 Black-box testing

Black box testing [12] (also called functional testing) is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. It bears the following advantages:

- the tests are done from the user's point of view and could help in exposing possible discrepancies in the specifications;
- a tester does not need to know the programming language in which the application has been implemented;
- the tests can be separated from the developers resulting in objective perspective in the testing procedures, increasing the testing efficacy;
- as soon as the specification is completed, the test cases could be designed;
- efficient for large code segments;
- access to the source code is not required.

The Black-box testing presents the following disadvantages:

- because of the complexity, only a few possible input sets are tested and many program paths will be untested (i.e. small code coverage).
- when the specification and documentation is incomplete or is not clear enough, it is difficult to design optimal test cases.
- redundancy of the tests if a system designer or a system developer has already run the test case in a different test level.

2.1.1.3 Grey-box testing

The main purpose of this testing approach is to search for defects and errors caused by the wrong structure or improper usage of applications. A grey-box tester has partial knowledge of the internal structures for purposes of designing proper tests and execute those tests on the black-box or user level. For designing correct and efficient tests, both a high-level specification and a detailed documentation of the application are needed [13].

Grey-box testing presents the following advantages:

- wherever it is possible it combines black-box benefits with white box benefits;
- the testers are not dependant on the source code only, they rely on the functional specification and the interface definition;
- thanks to the information available for the tester, better test scenarios especially around communication protocols and data type handling could be developed;
- the tests are done from the point of view of a user, neither the designer nor the developer.

Grey-box testing exhibits the following disadvantages:

- code coverage is limited because usually the testers do not have access to the whole source code.
- redundancy of the tests if the system designer or the system developer has already run the test case in a different test level.
- many parts of the complex system could stay untested because it is unrealistic to test all of the possible inputs in reasonable time.

2.2 Integration testing

Integration testing occurs after unit testing. The integration testing takes several modules or units, combines them into larger groups, sometimes called aggregates. The testers have already tested the individual components and the main purpose of the integration testing is to test the groups of units to see whether they properly interwork. The interfaces between units are tested without neglecting integration of the components and the hardware, the components and the operation system, or the components and the interface of any external system or application is tested as well. There are four approaches towards effective integration testing [14] :

- Bottom-up integration testing
- Top-down integration testing
- Big-bang integration testing
- Sandwich integration testing

2.2.1.1 Big bang testing

This approach is usually employed when the target of the test is the system as a whole; most of the components, if not all them, are tested at once. It is time-saving method in integration testing but there could stay some errors or defects untracked since not all the interfaces between the components are tested. The executed test cases and their results and outputs must be recorded properly to make the integration testing process clean and tabular.

2.2.1.2 Top-down testing

This testing approach tests the highest-level modules first and then lower level modules are tested step by step, until the tests hit the smallest possible division. This approach is usually chosen when it comes after top-down development of the system. Stubs are important in the top-down approach; a stub is a program or a method that simulates some functionality of lower-level modules. Since in top-down integration testing a higher-level is tested first, sometimes they need functionality of the lower-level modules which are not prepared for testing or which are not developed yet. The stub can provide an expected response to a request, or simulate some activity which is requested by the higher-level module.

2.2.1.3 Bottom up testing

This is an approach to integration testing where the lowest-level components are tested first, then they are combined and used for the testing of the higher-level components. The process is repeated until the component at the top of the hierarchy is tested. This approach helps in determining the completeness of software development processes, especially the percentage progress. When all the modules from the lower layer are ready, they get integrated and tested. To efficiently apply this testing approach, all components should be ready and prepared for the integration approximately at the same time. In such cases, a driver is necessary. Similar to a “reverse” stub, the driver is a method that passes some test cases to the lower-level modules or subsystems. This means that the driver simulates the higher integration layer that might not be implemented yet.

2.2.1.4 Sandwich testing

This approach combines top-down testing with bottom-up testing. There are no strict rules where the developers should start with development. This creates a system with missing components at various levels. Stubs and drivers are used where needed in order to allow test execution. Usually, less stubs and drivers development is required compared to pure top-down and bottom-up testing methods

2.3 Development Guidelines

To better organize the development activities conducted in the context of delivering the required functionality, we propose a preliminary plan for guidelines to be respected during the development and integration phases to facilitate partners’ interaction, interworking and facilitate code efficiency. In the following, such guidelines are discussed.

2.3.1 Guidelines for development

Since the development of the various software components of success will be performed by numerous, geographically dispersed partners and teams of the success consortium, efficient communication among them as well as comprehension of affected or affecting code segments/functionalities are all vital for optimizing integration in terms of efficiency and effectiveness. In the following paragraphs, the success development guidelines are presented in the form of rules that will ensure consistent and standardized (project-wide) coding practices. For a full list of good practices during code development, please refer to [15].

2.3.2 Design Patterns

Software design patterns are defined as general reusable solutions to commonly occurring problems, within a given context. They do not imply designs which can be directly transformed into code but, rather, constitute a formal way of documenting the solution to a known specific problem. Developers within success are encouraged to use design patterns, wherever applicable. Through design patterns success developers can share a standard terminology for a common problem, facilitating communication among them and boosting scalability and code effectiveness. Moreover, specified as common solutions, design patterns have been evolved by developers over time, so that they describe best practices in common problems.

Granted that success is promoting the enablement of real time meters and particularly the large-scale deployment of PMUs (through NORM), the amount of information delivered to the Utilities is expected to be significantly increased, implementing the “Big Data” paradigm. To this end, the success platform should be able to cope with Big Data streams, exhibiting linearly scalable performance, adapting to the power network needs. Though the Resilience by Design approach followed by success and developed in the context of WP2 is guiding the architectural design of the project towards this direction, popular cloud computing-oriented design patterns should be applied in conjunction, to ensure scalability at any time. Popular relevant design patterns are presented in [16] [17] [18].

2.3.3 Code comments

success developers are encouraged to consider the following principles while documenting their code:

- Comments should precede the code they refer to. During development, additions and modifications on the existing code could result in displacement of comments about the referred code. Developers should take care of their code structure and their comments placement, after code updates.
- Comments should be consistent with the code they refer to. success developers are encouraged to provide consistent documentation of their code.
- Comments should be short and concise, so that their effectiveness and readability does not get compromised.
- Comments should consist of the author name and date, so that potential incompatibilities can be reported to the original developers.

2.4 Integration Guidelines

In this chapter, preliminary guideline for facilitating integration activities are overviewed.

2.4.1 Respect over Interfaces and Data Models

A major principle for effective code integration relates to interfaces and data models. Data models define classes coupled with allowed methods, their signatures and parameters. On the other hand, interfaces specify the communication among the software components, defining the information flow, the interacting entities, their role in this interaction and the way this interaction is realized. In the same way well-defined data models and interfaces specifications may facilitate and actually enact components’ interaction, and not respected specifications may result in significant incompatibilities and lack of interaction. As most success developments will be conducted iteratively in three phases, enhancements and modifications of existing code will

necessarily take place. Hence, data models and interfaces may need to be modified, as well. success developers will respect specified interactions, data models and processes, to the extent possible. In the case of mandatory changes over specified processes, success developers are required to concretely specify the changes required, communicate them to the developing teams and include them in the revised version of the success architectural components.

3. Test plan for individual WP3 components

The goal of this chapter is to provide a preliminary strategy and plan that will be followed in the realization of the individual single component testing the belong to WP3, namely NORM and the DSOSMC.
This plan will subsequently feed the certification process for the NORM and the DSOSMC. The certification is an important part of the implementation of the SUCCESS components, because it confirms the seamless security and interoperability under normal and adverse conditions. The certification process to be followed within SUCCESS is conceptually described in D4.7 [2] and can be implemented for the WP3 components as well. Through the integration and validation plan, the necessary features of the components regarding security and interoperability will be pinpointed and test plans for a number of test cases will be defined. These attributes will provide an initial approach to the feature catalogue and the respective certification test plans.

To that end, we:

- introduce a common formalization list of terms for the well-defined definition of tests.
- briefly introduce the protocols that will be used between DSOSMC and NORM
- define the typology of the features that will be tested during the individual components testing
- define a preliminary plan describing the component testing of NORM and DSOSMC.

Individual module testing refers to the testing of the individual system components with respect to the requirements. After all modules are verified, then the integration will begin and when it is completed, a final set of tests will be performed.

3.1 Terms definition

It is useful to underline that to provide a common methodology for testing the main outcomes of WP3, a common definition of terms is needed. Since the devices, functions and systems under test do not represent a single field of technology, only technology-specific industrial grade testing procedures and terms have been adopted regarding the NORMs. The following definitions were collected and developed considering the state of the art in software, smart grid and system integration testing, especially with respect to the IEEE 829 Standard for Software Test Documentation [19] [17].

Table 1: Integration and testing definitions.

Term	Definition
(System integration test) Level	The amount of system layers which are included in a system integration test case minus one.
Device under test	A product which is verified by a certain test case. It is part of the test environment.
Expected results	A description of the status of the test environment after a test case was carried out and pass criteria have been met.
Features (not) to be tested	A list of product requirements or specifications which are (not) covered by a certain test case
Pass/fail criteria	A definition how to judge or measure if a product under test is conform to specifications and requirements that shall be validated by a certain test case

Retesting	Re-execution of a test case that previously returned a “fail” result, to evaluate the effectiveness of intervening correction actions.
Subsystem acceptance criteria	Conditions to be fulfilled by a subsystem for including it into the system integration test. Conditions- should include the availability of testing protocols for standalone subsystem tests. Also, subsystems should have similar level of maturity.
Suspension criteria	A description of conditions which indicate that the test was carried out incorrectly or that any situation was produced which renders the testing results unusable, making test continuation pointless and requiring the test to be halted and restarted
System integration test	A test designed to verify that a system made up of two or more interacting products (subsystems) conforms to system-wide specifications and requirements. The device under test is the system itself. It is especially designed for finding inconsistencies which emerge only through the subsystem interaction. The system integration test plan may define partial system integration tests which allow for adding subsystems subsequently.
System layer	A group of one or more subsystems which is defined prior to the system integration test. According group definition for a given system should be used for all system integration test cases.
Test analysis	Elaboration about why a test result emerged. May also include a conclusion about what the test result implies for the future work.
Test case code	An identifier for a test case which is unique throughout the project, e.g. SC13:003.
Test case specification	Documentation of one or more test cases
Test coverage	A list of product requirements or specifications which are verified by a test plan
Test data	Data created or selected which is needed for executing one or more test cases. May be defined in the test case specification.
Test environment	A list of all elements (software, hardware, information, external conditions) needed to carry out a test case, including the device or system under test and all elements needed to judge the test outcome.
Test environment set-up process	A list of actions needed for establishing and maintaining a required test environment
Test execution	The action of carrying out the testing procedures for a given test case.
Test group	A collection of test cases which share at least one defined criterium. E.g. all test cases which relate to cyber security testing might be defined to make up a test group.
Test method	A general definition of testing procedures and test environment for a test plan
Test plan	A strategy or list of tasks used to verify that a product conforms to design specifications and product requirements.
Test preparation	A definition of steps which are needed to prepare a test environment for test execution
Test protocol	A summary of the test results of all test cases defined in a test plan. It may also contain the test analysis for said test cases.
Test requirements	A definition stating the status of the test environment which is needed for carrying out a specific test case or a test group. Ideally, it is also stated how it can be checked if the test environment is ready for test execution.
Test responsibilities	A definition stating which persons or organizations are needed for the test. It may also include an assignment of tasks to those persons or organizations.
Test result	Indication of whether a specific test case has passed or failed. May also include any data that has been obtained through execution of the test case.
Testing	Set of activities conducted to facilitate discovery, validation and/or evaluation of properties of one or more test NOBEL GRID components
Testing procedures	A specific list of steps which are needed to carry out a test case

3.2 Test Criteria

The tests can also be divided following a certain number of criteria. A first preliminary list has selected a set of these common criteria providing a classification of tests types which are reported in **Fehler! Verweisquelle konnte nicht gefunden werden.**. This Criteria will allow a classification on the features to be tested in the next deliverables versions.

Table 2: Tests Classification

Test group	Common Criteria
Communication	Tests regarding compliance to norms, regulations, standards, policies or laws
Hardware	The feature under test is provided by hardware only
Functionality	The feature under test is a function provided by a combination of software, hardware and communication
Security & Robustness	Tests related to vulnerabilities of software, hardware or communication to malicious attacks aimed at the NORM; tests regarding sensitivity of the SMG towards adverse external conditions (e.g. temperature, mechanical stress or similar)
Provisioning	Test related to installation, updating, maintenance, or repair of hardware or software
Software	The feature under test is provided by software only
Compliance	Tests regarding compliance to norms, regulations, standards, policies or laws
Interoperability	The feature under test relates to interoperability of two or more software or hardware components

3.3 Test case specification

Test cases will use a template sheet as shown in **Table 3** In this preliminary version will be provided only this template but in the next deliverables D3.11 [3] and D3.12 [4] this specification will be extended with details of the test cases.

Table 3: Template

Name	*A test case code and name which is unique to the project. Should include one-line test description.*	**Locations**	*The places where the test will be executed*
Product under test	*The device or system under test*	**Resp.**	*The partner responsible*
Product requirement	*The requirement, use case, or certification rule which is validated by the test case.*		
Test environment	*List of elements needed for the test execution. May refer to a general test environment.*		
Features under test	*List of features under test*		
Features not under test	*Optional*		
Preparation	*Short list of preconditions which the test environment has to meet before test execution*		
Dependencies	*Optional. List of codes of the test cases which need to be verified before this test*		

	case can be started. This shall reflect that the test case at hand depends on features previously tested by other test cases.
Steps	*Testing procedures, i.e. list of actions needed for test execution*
Pass criteria	*Expected (measurable) results, allowing to unambiguously judge if the test is passed or not passed (i.e. the product requirement was validated or not validated).*
Suspension criteria	*Optional. Conditions under which continuation of the test is considered pointless because testing results would be invalid.*

3.4 NORM

The device that under test is the NORM. A detailed description of the NORM in Figure 1 can be found in [7] The NORM is composed of a Smart Meter Gateway (SMG), a low cost Phase Measurement Unit (PMU) and a board with a Physically Unclonable function (PUF). The metrological part connected to NORM is responsible for recording the metrological data but has limited functionalities. The SMG overcomes this functional limitation by providing advanced functionalities that support the future evolution of the smart grid and energy services. The SMG has the possibility to communicate with both home area networks (HAN) and public networks through internet to provide its advanced functionalities.

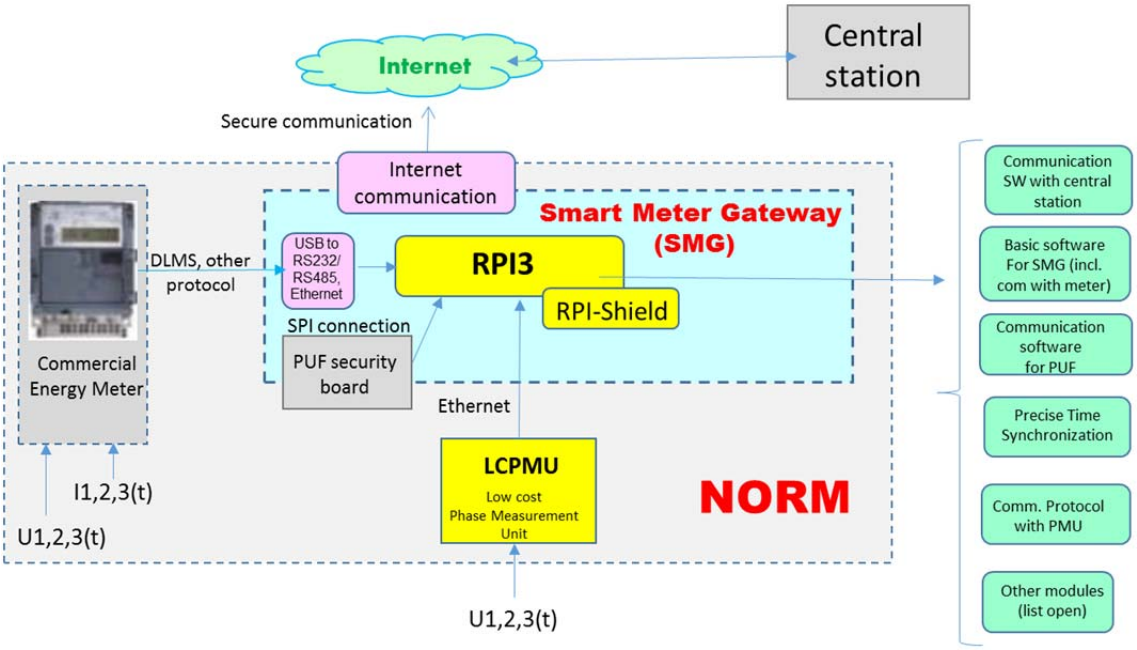

Figure 1 : NORM

3.4.1 Test Plan

The device that will be tested is the NORM, which is under development in T3.4. The following steps are used to describe a preliminary of test plan that will be further specified in subsequent deliverables:

- Review the project use cases and standards related to smart metering, smart grid and energy services.
- Define the features to be tested and classify them into test groups
- Detailed test cases for validation of named features
- Execute the test cases

- Document the test protocols

The features to be tested are defined in preliminary version following what needs to be validated in order to guarantee that the NORM is operating according to the standards and fulfilling the project requirements and use cases. Test cases define the testing procedure in a controlled environment in the laboratory of a technical project partner.

3.5 DSO Security Monitoring Centre

The DSOSMC is a central component in the success overall information security architecture (Figure 2), undertaking security monitoring and actuation tasks at NaN level. The core operations performed by the DSOSMC are as follows:

- Interact with NORM devices, receive metering and management information and have access to basic control actions;
- Identify and mitigate (apply countermeasures) attacks against the Smart Grid at NaN level;
- Provide aggregated security information to the Utilities, in the form of KPIs;
- Provide a dashboard to Utilities from where they will be able to overview the status of their Smart Grid network of interest and interact with the attack identification and mitigation mechanisms of the DSOSMC.

Further details on the DSOSMC specifications are provided in [8].

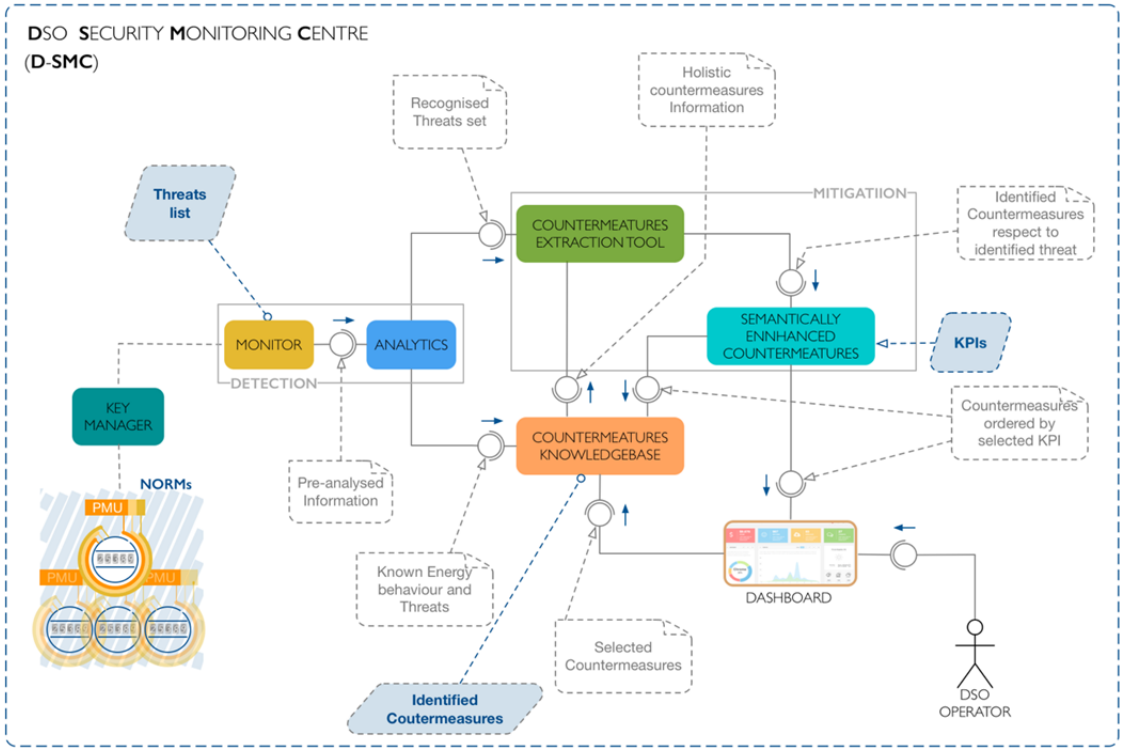

Figure 2: DSO Security Monitoring Centre conceptual architecture.

3.5.1 Test Plan

The DSOSMC implementation is performed in the context of T3.3. In the same way to §3.4.1, the following steps are used to describe a preliminary of test plan that will be further specified in subsequent deliverables

:

- Review NORM specifications and exposed interfaces;
- Define the features to be tested and classify them into test groups;
- Detailed test cases for validation of named features;
- Execute the test cases;
- Document the test protocols.

The features to be tested are defined in preliminary version following what needs to be validated in order to guarantee that the DSOSMC is operating according to the standards and fulfilling the project requirements and use cases. Test cases define the testing procedure in a controlled environment in the laboratory of a technical project partner.

4. Interaction between NORM and DSOSMC

4.1 Data Flow

DSOSMC will get real time measurement in push mode from API on NORM side .

Data Flow	Connection Type	API Protocol	Data Type	Comments
Near real time measurements	TCP/IP	REST, MQTT	JSON/XML	The DSOSMC will the real-time information on various metrics that are measured from the smart meter and are proxied or processed by NORM, such as voltage dips, angles between voltage samples, average network traffic etc. More details will be presented in D3.7 [7]
Management commands	TCP/IP	REST	JSON/XML	Commands sent to the NORM from the DSOSMC to control its operation, e.g. refresh the PUF active challenge, restart a specific service etc.
Local Security Agent Measurements	TCP/IP	REST/MQTT	JSON/XML	Calculated measurements that are not privacy-breaking, e.g. Voltage-related (and voltage diffs) information, open connections, interface traffic statistics, Credentials / Challenge / Response updates, role based access control statistics, tampering detection reports

Table 4: Data Flow

4.2 Installation details

Integration could be realized between the installation of the real DSOSMC on Engineering premises and either with a simulated NORM either a real installation depending of the time of the project.

4.3 Time schedule

Following a plan for the time schedule of test and integration of NORM and DSOSMC . Both individual test and integration will follow in parallel and the due handover are summarized in Table 5: Integration time planTable 5.

Action	Test	Start Date Month	End Date Month
Individual test	DSOSMC Test	12	23
Individual test	NORM Test	12	23
First Integration test round	First round of integration between DSOSMC and NORM	12	16
Second Integration test round	NORM and DSOSMC	16	23
Third Integration test round	Second round of integration test between DSOSMC and NORM	23	24

Table 5: Integration time plan

5. References

[1] success, “Deliverable D4.4: Description of available components for SW functions, infrastructure and related documentation v1,” 2017.

[2] success, “Deliverable D4.7 : Integration and Validation Plan. Test and certification specifications v1,” 2017.

[3] success, “Deliverable D3.11: Integration and Validation Plan. Test and certification specifications v2,” 2017.

[4] success, “Deliverable 3.12 : Integration and Validation Plan. Test and certification specifications v3,” 2018.

[5] success, “Deliverable D4.8 : Integration and Validation Plan. Test and certification specifications v2,” *2017.*

[6] success, “Deliverable 4.9 : Integration and Validation Plan. Test and certification specifications v3,” *2018.*

[7] success, “Deliverable D3.7 : Next Generation Smart Meter V1,” 2017.

[8] success, “Deliverable D3.4: Information Security Management Components and Documentation V1.0,” 2017.

[9] M. J. a. M. L. S. Harrold, “An incremental approach to unit testing during maintenance.,” *Software Maintenance,,* 1988.

[10] K. Beck, Test-driven development: by example, Addison-Wesley Professional, 2003.

[11] T. Ostrand, White-Box Testing, Encyclopedia of Software Engineering, 2002.

[12] B. Beizer, Black-box testing: techniques for functional testing of software and systems., John Wiley & Sons, Inc., 1985.

[13] M. E. a. F. K. Khan, “A comparative study of white box, black box and grey box testing techniques.,” *International Journal of Advanced Computer Science and Applications (IJACSA),* 2012.

[14] H. K. a. L. W. Leung, “A study of integration testing and software regression at the integration level.,” *Software Maintenance, 1990, Proceedings., Conference on.IEEE,* 1990.

[15] D. Spinellis, “15 Rules for Writing Quality Code,” 9 06 2014. [Online]. Available: http://www.informit.com/articles/article.aspx?p=2223710. [Accessed 17 09 2014].

[16] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel, A Pattern Language, Towns, Buildings, Construction, New York: Oxford University Press, 1977.

[17] “. 2. S. T. The International Software Testing Standard. [Online]. Available: http://www.softwaretestingstandard.org/. . [Accessed 10 September 2014].

[18] “Big Data Patterns,” [Online]. Available: http://www.bigdatapatterns.org/. [Accessed Jan. 2017].

[19] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Software_test_documentation. [Accessed 10 January 2017].

[20] “Cloud Patterns,” [Online]. Available: http://cloudpatterns.org/. [Accessed Jan. 2017].

[21] success, “Deliverable D3.4: Information Security Management Components and Documentation v1,” 2017.

[22] FIWARE, “Cygnus,” [Online]. Available: https://github.com/telefonicaid/fiware-cygnus. [Accessed Jan 2017].

[23] NOBEL GRID, “D1.1-Distribution grid and retail market requirements definition,” 2015.

[24] NOBEL GRID, “D1.2-Distribution grid and retial market scenarios and use cases definition,” 2015.

[25] European Union, “RECTIVE 2014/32/EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 26 February 2014 on the harmonisation of the laws of the Member States relating to the making available on the market of measuring instruments (recast),”

Offical Journal of the European Union, 2014.

[26] E. Union, “RECTIVE 2014/32/EU OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 26 February 2014 on the harmonisation of the laws of the Member States relating to the making available on the market of measuring instruments (recast),”,” *Official Journal of the European Union,* 2014.

[27] “Protection of personal data,” [Online]. Available: http://ec.europa.eu/justice/data-protection/.

[28] “REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL,” [Online]. Available: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679.

[29] IEC, “62056-6-2:2013,” 201. [Online]. Available: https://webstore.iec.ch/publication/6410. [Accessed 20 12 2016].

[30] N. a. B. J. C. Saxena, “State of the art authentication, access control, and secure integration in smart grid,” *Energies,* 2015.

[31] S. Feuerhahn, M. Zillgith, C. Wittwer and C. Wietfeld, “Comparison of the communication protocols DLMS/COSEM SML IEC 61850 for smart metering applications,” in *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on. IEEE*, Bruxelles.

[32] “ OpenADR and Cyber Security,” [Online]. Available: http://www.openadr.org/cyber-security. [Accessed 20 12 2016].

6. List of Abbreviations

BR-GW	Breakout Gateway
BSF	Bootstrapping Server Function
CA	Certificate Authority
DCS	Data Centric Security
DEMS	Decentralised Energy Management System
DoS	Denial of Service
DDoS	Distributed Denial of Service
DE-SMIS	Distributed instance of European Security Monitoring and Information System
DSO	Distribution System Operator
DSOSMC	DSO Security Monitoring Centre
ESMC	Pan-European Security Monitoring Centre
E-SMIS	Security Monitoring and Information System
NAN	Neighbourhood Area Network
NORM	New-generation Open Real-time Smart Meter
PMU	Phasor Measurement Unit
PUF	Physically Unclonable Function
SCADA	Supervisory Control and Data Acquisition