LOCARD

# DELIVERABLE

# D5.1 Established Test Environment

| | | |
|---|---|---|
| ***Project Acronym:*** | LOCARD | |
| ***Project title:*** | Lawful evidence collecting and continuity platform development | |
| ***Grant Agreement No.*** | 832735 | |
| ***Website:*** | http://locard.eu/ | |
| ***Contact:*** | info@locard.eu | |
| ***Version:*** | 1.0 | |
| ***Date:*** | 30 April 2020 | |
| ***Responsible Partner:*** | ICO | |
| ***Contributing Partners:*** | ICO, IMC, MOT | |
| ***Reviewers:*** | Constantinos Patsakis (ARC), Hugo Kerschot (VIC) | |
| ***Dissemination Level:*** | Public | X |
| | Confidential – only consortium members and European Commission Services | |



This project has received financial support from the European Union Horizon 2020 Programme under grant agreement no. 832735.

| Revision | Date | Author | Organization | Description |
|---|---|---|---|---|
| **0.1** | 09/12/2019 | Yiannis Perros | ICO | Tentative ToC/structure |
| **0.2** | 13/01/2020 | Yiannis Perros, Fotis Kouretas, Miltiadis Anastasiadis | ICO, IMC, MOT | Updated Version |
| **0.3** | 31/01/2020 | Yiannis Perros, Fotis Kouretas, Miltiadis Anastasiadis | ICO, IMC, MOT | Updated Version |
| **0.4** | 28/02/2020 | Yiannis Perros, Fotis Kouretas, Miltiadis Anastasiadis | ICO, IMC, MOT | Updated Version |
| **0.5** | 09/03/2020 | Lelia Ataliani | ICO | Version for review |
| **0.6** | 26/03/2020 | Yiannis Perros, Fotis Kouretas, | ICO, IMC | Updated Version |
| **0.7** | 14/04/2020 | Yiannis Perros, Lelia Ataliani, Fotis Kouretas, Miltiadis Anastasiadis | ICO, IMC, MOT | Updated Version |
| **0.8** | 21/04/2020 | Constantinos Patsakis, Hugo Kerschot | ARC, VIC | Review remarks |
| **1.0** | 29/04/2020 | Yiannis Perros | ICO | Final version |



Every effort has been made to ensure that all statements and information contained herein are accurate, however the LOCARD Project Partners accept no liability for any error or omission in the same.

# Table of Contents

# 1 Executive Summary

This deliverable establishes the standards, software industrial tools for collaborative development and sets up the corresponding test environment, which supports the continuous integration and quality assurance of all LOCARD modules as well as final integration.
This is a detailed document that describes the test strategy, objectives, roles and deliverables planned for testing. The Test Plan helps us validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process which is minutely monitored and controlled by the test and quality assurance team.
Finally, a list of tools is presented that enable seamless collaboration of the technical team, all of which are installed on Motivian infrastructure on secure servers, where the whole development and testing of LOCARD modules and LOCARD integrated system will take place.

# 2 Introduction

The Test Plan is designed to prescribe the scope, approach, resources, and schedule of all testing activities of the LOCARD project. The plan identifies the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.

After the user acceptance phase and subsequent corrective actions, it will ensure that the system is working properly and has all the required functionalities to support the required by the project functionality. Unit tests will assess automatically whether individual modules perform the tasks they are assigned with correctly.

During this phase, penetration and usability – accessibility tests are going to be performed. Penetration testing will be conducted to ensure that the system does not have known security gaps and vulnerabilities. Usability and Accessibility testing are conducted to ensure that the system meets the specifications initially set.

## 2.1 Scope

Scope defines the features, both functional and non-functional requirements of the software that **will be** tested. All features of LOCARD project which have been defined in software requirement-specs will be tested both ad hoc and within an integrated environment.
LOCARD modules that will be tested are:

- Crowdsource Intelligence
- Data Collection modules
    - *The Intelligent Crawler*
    - *The Investigator's Toolkit*
- Storage Manager
- Blockchain Manager

- User and Identity Manager
- Intelligence Engine
- Deviant Patterns repository
- Connector
- Reporting Engine
- Alert Engine
- Communication Engine
- Trusted Execution Environment (TEE)
- LOCARD Core Orchestrator Bus
- UI/UX Interface and Portal

## 2.2 Quality Objective

The test quality objectives are to **verify** the operational and functional specifications of all LOCARD modules after thorough testing prior to the delivery of the system to the end users. Quality assurance is based on ISO standards 9001 and 27001 that the technical partners adhere to, and will be followed and monitored ensuring the integration of all LOCARD modules to **guarantee** all these operations can work **smoothly** in a real working environment. Moreover, the technical partners will test the feasibility of the test collaborative tools mentioned below and their merit in contributing to the development process.

The quality management of the envisaged project is a horizontal task covering all aspects of the contract. Due to the nature of the services that the consortium will be called to carry out, the quality management is of paramount importance in mastering a complex structure, which must be available and ready to undertake ad hoc working requests in a diversity of systems, environments and locations. The primary objective of the quality management is to undertake the monitoring of the development and deployment and ensure that all the targets set forth are met.

The consortium fully appreciates the significance of the envisaged activities. It will therefore place the utmost importance in the implementation of a quality management and control organisation, which will guarantee the uninterrupted and efficient service completion. The shaping of our approach to quality is being based on a number of key principles, which are given below.

- **Proactive approach.** First and foremost, the proactive approach is concerned with the project's technical team. Its members are well educated and empowered so that they can actively seek opportunities to improve quality, based on early indications for potential problems obtained from the continuous monitoring of the project quality. Our approach is to seek measures that will solve problems before they even appear.
- **Continuous process improvement.** Quality improvement will be a continuous activity, aiming at having better process effectiveness and efficiency. It will be necessary to constantly evaluate working and management practices, guidelines, modes of operations, etc., with a view to improve the resulting quality.
- **Measuring satisfaction**. Satisfaction is the result of a number of positive and negative factors that are experienced by the users. For a successful project, it is essential that the vast majority of users are highly satisfied. Closely working with wp6 – where satisfaction will be measured - and getting feedback iteratively as the platform integration progresses, the technical team will enhance platform useage and eliminate errors so that end users are satisfied.

## 2.3 Roles and Responsibilities

| | | |
|---|---|---|
| **1.** | Test Manager | Manage the whole project testing<br><br>Define project directions<br><br>Acquire appropriate resources from technical partners |
| **2.** | Test | Identifying and describing appropriate test techniques/tools/automation architecture Verify and assess the Test Approach Execute the tests, Log results, Report the defects.<br><br>Outsourced members |
| **3.** | Developer in Test | Implement the test cases, test program, test suite etc. |
| **4.** | Test Administrator | Builds up and ensures test environment and assets are managed and maintained<br><br>Support Tester to use the test environment for test execution |
| **5.** | Quality Assurance Manager | Take in charge of quality assurance<br><br>Check to confirm whether the testing process is meeting specified requirements |

# 3 Test Methodology

## 3.1 Overview

Software Testing Methodology is defined as a set of strategies and testing types used to certify that the Application / system developed and commissioned under Test meets the project and end users' expectations. Test Methodologies include both functional and non-functional testing to validate this. Examples of Testing Methodologies are Unit Testing, Integration Testing, System Testing, Performance Testing etc. Each testing methodology has a defined test objective, test strategy, and deliverables. Testing Methodologies could refer to Waterfall, Agile and other QA models as against the above definition of Testing Methodologies. In this

project we will adopt the Agile testing methodology. In Agile methodology, software is developed in incremental, rapid cycles as presented later in section 7.

Testing strategy will consist of the Incremental testing type meaning that every release of the project is tested thoroughly. This ensures that any bugs in the system are fixed before the next release. Moreover, in regard to the testing and validation, the data used will be synthetic.

## 3.2 Testing Process

Testing procedures will be defined and followed to verify and validate any software developed in WP5. Verification is the process of evaluating the system or a system component to determine whether the products of the development phase satisfy the conditions imposed at the start of the phase (i.e. ensure that the development team builds the system right). Validation is the process of evaluating the system or a system component during or at the end of the development process to determine whether it satisfies specified requirements (i.e. ensure that the development team builds the right system).

The goals of testing and acceptance will be to:

- Find errors/defects in the systems as efficiently and timely as possible;
- Evaluate the various aspects of the systems or a systems component to ensure it performs as expected;
- Reduce both the number of defects delivered and the risks associated with those defects;
- Establish confidence that the new system does what it is supposed and expected to do by the LOCARD Consortium.
- Reduce the overall risk within the project.

## 3.3 Test Types

The following test types will be executed during the project according to the type of development and/or maintenance activity.

### 3.3.1 Functional Tests

The goal of functional testing is to verify that the application functionality requirements are met according to the Software and System Specification - as presented in D3.4 and D3.5 - and the related user cases in D3.3.

Functional tests will be based upon black box techniques. Black box testing is concerned only with testing the behaviour of the system from an external point of view. It verifies the acceptance/input, processing, and retrieval/output of data according to system specifications without considering the internal application structure. Black box testing is testing against the software requirements specification and the related test cases and will discover faults of omission, indicating what part of the specification has not been fulfilled.

Typically, functional testing involves the following steps:

- Identify functions that the software is expected to perform and document them in test cases and test scenarios.
- Create input data based on the function's specifications.
- Determine the output based on the function's specifications.
- Execute the test case.
- Compare the actual and expected outputs.

### 3.3.2 Unit Testing

The unit testing will involve testing of the individual system's software units or groups of related units. Each unit of code will be evaluated on how well it meets the requirements for which it was designed. Timing, memory, accuracy in producing numerical and logical results will be tested as well as the preparation of input and output required for validating program logic, syntax, and performance requirements. The unit testing conclusions should clearly state if the module under test is ready for integration with the rest of the modules, based on the number of bugs outstanding. After successful unit testing, link testing will be conducted to test the result of putting together the modules and sub-modules that successfully passed the unit testing. Link tests will be designed to provide assurance that dependent groups or sequences of the system's software units (module clusters, program clusters, remote dialogue responders) can communicate in the way intended. This is not a test of functionality, although obviously some functionality will be tested in the process of testing linkage.

### 3.3.3 Integration Testing

Integration testing focuses on the testing of APIs and web services provided by the system to external systems. Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. Activities during this phase will include: (a) Generate Integration Test Data - Building actual data into files, databases or test harnesses, using values stated in specifications. (b) Conduct Integration Tests - Performing steps according to scripts, registering timings, printing diagnostics, and noting all significant events. (c) Analyse Integration Test Results - Revisiting test outputs systematically, checking accuracy and consistency, and identifying non-conformity. (e) Support Corrective Actions - Diagnosing and correcting faults, re-testing according to standards completing the relevant documentation under configuration management. (f) Integration Test Report - Evaluation of the complete test process and production of the Integration Test File that will contain information on test log and results analysis, error correction and re-test and final reporting.

### 3.3.4 Security tests

Security tests will be performed on an application-level, with the goal to verify that users can access only those functions / data for which their user type is provided permissions, and on a security level, with the goal to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration or usage. The application-level security testing will also cover the features foreseeing data encryption and user password policy as described in OWASP Top 10 security risks.

A security testing methodology will be utilised in order to identify, describe, classify and report the vulnerabilities of the system. Information about policies, standards and the documentation covering the products to be tested, threats against them and security controls will be gathered and used as input for test planning.

The security testing techniques and activities will be decided for each category of vulnerability. For example, testing authentication could include testing activities like credentials transport over an encrypted channel; user enumeration; guessable user account; brute force; bypassing authentication schema; vulnerable remember password and password reset; logout and browser cache management; and more. As another example, testing authorisation could include testing activities like path traversal; bypassing authorisation schema; privileged escalation.

### 3.3.5 Recovery Testing

Recovery testing is a system test that focuses the software to fall in a variety of ways and verifies that recovery is properly performed. If it is automatic recovery then re-initialization, check pointing mechanisms, data recovery and restart should be evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

A variety of hardware, software or network malfunctions might be caused on the LOCARD system and verify that the software can successfully recover from them without loss of data or data integrity. Recovery processes will be invoked, and the system will be monitored and inspected to verify proper recovery of data and data integrity.

### 3.3.6 User Interface (UI) Tests

UI tests verify the user's interaction with the software. The goal of UI Testing is to ensure that:

- the UI provides the user with the appropriate access and navigation through the functions of the application and presents to the user all the data in the appropriate language;
- the elements and components of the UI conform to commonly used standards;
- the colours and visual components follow accessibility guidelines.

An application may be built in total conformance with the software specifications; however, it may be ‘unusable’ when it lands in the hands of the end-users. In UI testing, the software is tested from a user point of view to prove/ensure that it is user-friendly and allows the users to achieve their goals. UI testing aims to ensure that the user interface design (layout and sequence, etc.) enables the execution of business functions as easily and intuitively as possible.

The idea behind UI testing is to have users perform the tasks for which the application was designed. If they cannot do the tasks or if they have difficulty performing the tasks, the User Interface (UI) is not appropriate and should be redesigned. The techniques to be applied for testing LOCARD UI include usability testing, expert reviews and tree testing.

### 3.3.7 Performance Tests

Performance tests measure response times, transaction rates, and other time sensitive requirements according to specifications. The goal of Performance tests is to verify and validate that performance requirements have been achieved. Performance tests have to be usually executed several times, each using a different "background load" on the system. Performance testing can also measure what parts of the system or workload cause the system to perform badly. In the diagnostic case, software engineers use tools such as profilers to measure what parts of a device or software contribute most to the poor performance or to establish throughput levels (and thresholds) for acceptable sustained response time.

Performance testing technology employs one or more workstations to act as injectors – each emulating the presence or numbers of users and each running an automated sequence of interactions (recorded as a script or as a series of scripts to emulate different types of user interaction) with the host whose performance is being tested. Usually, a separate workstation acts as a test conductor, coordinating and gathering metrics from each of the injectors and collating performance data for reporting purposes. The usual sequence is to ramp up the load – starting with a small number of virtual users and increasing the number over a period to some maximum. The test result shows how the performance varies with the load, given as number of users vs. response time. Various tools are available to perform such tests. Tools in this category usually execute a suite of tests that emulate real users of the system. Sometimes the results can reveal oddities, e.g. that while the average response time might be acceptable, there are outliers of a few key transactions that take considerably longer to complete; this might be caused by inefficient database queries, etc.

The technical team will conduct the following:

- Analysis of the types of interaction that should be emulated and the production of scripts to do those emulations;
- Set up of a configuration of injectors/controllers;
- Set up of the test configuration (ideally identical hardware to the production platform), router configuration, quiet network, deployment of server instrumentation;
- Running the tests – probably repeatedly in order to see whether any unaccounted factors might affect the results;
- Analysing the results, either pass/fail, or investigation of critical path and recommendation of corrective action.

Performance testing can be combined with stress testing, load testing and soak testing in order to see what happens when an acceptable load is exceeded (does the system crash? How long does it take to recover if a large load is reduced? Does it fail in a way that causes collateral damage?).

**Load Testing**

The objective in load testing is to analyse the effect of a realistic number of users accessing an application. By load testing many problems may be identified before a large number of users use the application. The technique of load testing is to have the application accessed by many simulated users. This is done using testing applications (JMeter in LOCARD). Once the case is created in the testing tool, it may be triggered many times using different parameters, such as thread number, think time and loop count. The parameters

should be set in a way to simulate the effect of the real-world user environment on an application. Load testing should be performed on a dedicated machine or at a dedicated time. This permits full control and accurate measurements. The databases used for load testing should be either actual size or scaled equally.

**Stress Testing**

During stress testing a system is operated in a manner that demands resources in abnormal quantity, frequency or volume. The following types of tests are conducted during stress testing:

- Special tests designed to generate ten interrupts per second, when one or two is the average rate;
- Input data rates increasing by an order of magnitude to determine how input functions will respond;
- Test Cases requiring maximum memory or other resources;
- Test Cases causing excessive hunting for disk-resident data;
- Test Cases causing thrashing in a virtual operating system.

### 3.3.8 Factory Acceptance Tests

Before issuing a software release, testing, analysis and reporting of results will be performed by the technical team. The testing activities that are foreseen in a FAT are the following: Functional Testing, Usability Testing, Security Testing, Performance Testing.

### 3.3.9 Penetration Testing

Penetration tests are a great way to identify vulnerabilities that exist in a system or network that has existing security measures in place. A penetration test usually involves the use of attacking methods conducted by trusted individuals that are similarly used by hostile intruders or hackers. Depending on the type of test that is conducted, this may involve a simple scan of an IP address to identify machines that are offering services with known vulnerabilities or even exploiting known vulnerabilities that exist in an unpatched operating system. The results of these tests or attacks are then documented and presented as reports of the system and the vulnerabilities identified can then be resolved. A penetration test is basically an attempt to breach the security of a network or system and is not a full security audit. This means that it is no more than a view of a system's security at a single moment in time. At this time, the known vulnerabilities, weaknesses or misconfigured systems have not changed within the time frame the penetration test is conducted. Penetration testing is often done for two reasons. This is either to increase upper management awareness of security issues or to test intrusion detection and response capabilities. It also helps in assisting the higher management in decision-making processes. Penetration tests can have serious consequences for the network on which they are run. If it is being badly conducted it can cause congestion and systems crashing. In the worstcase scenario, it can result in the exactly the thing it is intended to prevent. This is the compromise of the systems by unauthorized intruders.

The penetration test starts by gathering all possible information available regarding the infrastructure and applications involved. This stage is paramount as without a solid understanding of the underlying technology involved, sections may be missed during the testing phase. The test will follow all the different phases described below:

- Testers will attempt to exploit all discovered vulnerabilities. Even if the exploitation fails, the tester will have a better understanding of the vulnerability risk
- Any information returned by checking for vulnerabilities, for example, programming errors, source code retrieval or other internal information disclosure, should be used to re-assess the overall understanding of the application and how it performs.
- If at any point during the testing a vulnerability is detected, which may lead to the successful compromise of the target or disclose business-critical information, the relevant contact person of the technical team will be contacted immediately and made aware of the situation and risks involved.

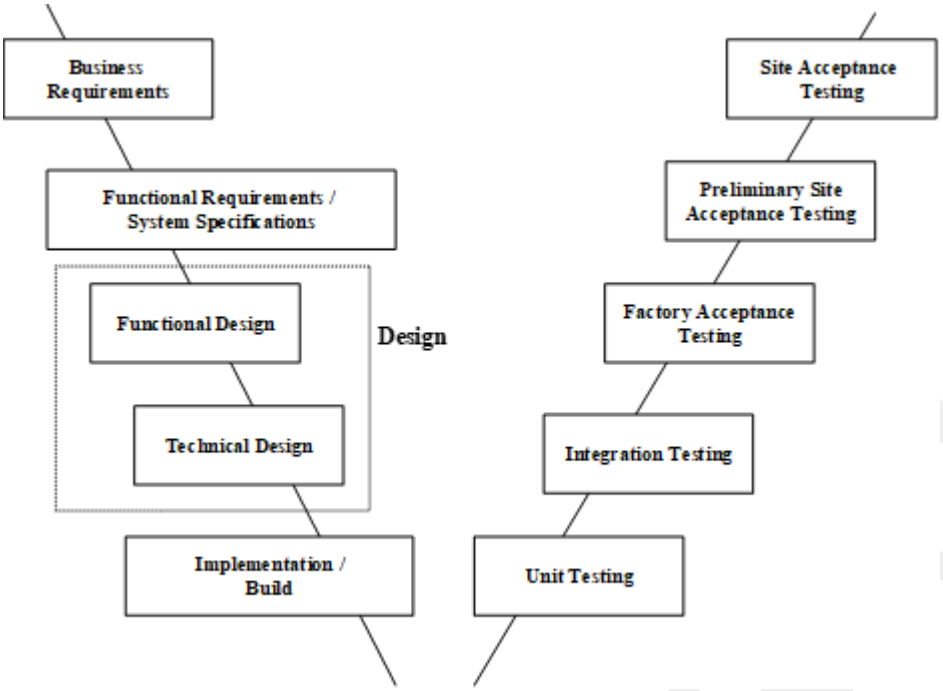
### 3.3.10 User / Site Acceptance Tests

Site Acceptance Testing (SAT) phase is initiated upon each software release and following the successful deployment of the system in the staging environment. End Users testers in LOCARD will conduct Site Acceptance Testing by executing the test scenarios and cases specified in the Test Plan. It is anticipated that all the test types of the FAT will also be conducted in the SAT. The technical team will assist LOCARD end users personnel for the execution of the SATs.

User acceptance testing (UAT) will be also conducted; it will be executed by end users with the support of the technical team that will plan and manage the entire process. End users may perform any additional tests considered necessary in order to fully ensure that the system performs in accordance to the specifications.

## 3.4 Testing Lifecycle and Deliverables

The Testing lifecycle will be closely related to and synchronized with the development/maintenance life cycle of the systems. The testing process will involve both the application software as well as the technical infrastructure (hardware, system software, etc). The V-shape relationship defined below reflects this interaction.




The Testing lifecycle will produce the following document deliverables:

- The Testing Strategy – it defines the overall testing approach for the system under development / maintenance; it includes methodology, acceptance criteria and other quality indicators;
- The Testing Plan – defines the steps to be used to test the system under development / maintenance including test scenarios, test cases and their related parameters (input data, expected results, environment, pass/fail criteria, importance);
- The Unit Testing Report – it describes the tests run during the unit testing of a (stand-alone) module or submodule of the system under development / maintenance; it also includes the results of the testing session and the conclusion;
- The Integration Testing Report – it describes integration tests (test scenarios and test cases) run during the integration of a number of modules or sub-module (eventually the complete set) of the system under development / maintenance; it also includes the results of the testing session and the conclusion;
- The FAT (System Test) Report – this document includes the description of the final test scenarios, test cases and tests run by the LOCARD technical team once the integration is finished successfully and the system is ready to be delivered; it also includes the results of the FAT session and the conclusion; the FAT session will be done on the technical team premises, more at a business / functional level and in a configuration and with data very similar to the production environment;
- The pSAT Report – this document contains the results of the pSAT Phase performed by technical team; it is made available to all concerned and / interested parties;
- The SAT Plan – defines the steps to be used to test the new system delivered to the end users including test scenarios, test cases and their related parameters (input data, expected results, environment, pass/fail criteria, importance);
- The SAT Report - this document includes the description of the test scenarios, test cases and tests run by the technical team once the FAT and pSAT are finished successfully and the new system is

delivered for acceptance; it also includes the results of the SAT session and the conclusions; the SAT session is done by the wp5 team in the wp5 leader's acceptance environment, more at a business / functional level and in a configuration and with data very similar to the production environment;

- The Certificate of Acceptance– this document is issued after a successful SAT session; it is addressed to the Consortium and represents the acceptance by the technical team of the system release delivered;

## 3.5 Acceptance Criteria

Acceptance Criteria will define, as briefly as possible, the rules the technical team will use to determine whether the Consortium's work has been successfully completed. Good Acceptance Criteria minimize subjectivity in deciding whether the work has been done and provide a starting point for test planning.

In general, Acceptance Criteria:

- Will be derived from goals and performance indicators. The highest-level system and business goals and performance indicators will be the starting point for identifying Acceptance Criteria. However, these goals and indicators should seldom be translated directly into Acceptance Criteria.
- Will be derived from agreed-upon quantitative requirements. All Acceptance Criteria will be based on the contractually agreed-upon terms. General and long-term business goals set at a high level will not be applicable unless they have been incorporated into the specification process and adopted as achievable in the specific release for which acceptance is being sought.
- Will relate to the essential "What" versus the arbitrary "How." Acceptance will focus on functionality and usability, rather than on the design approach per se. It is therefore important that the specification of requirements support the maximum degree of freedom available for the architecture and design.
- Will define a formal Pass/Fail process. Ultimately, the purpose of acceptance testing is to formally acknowledge completion of work. It will not be used to evaluate how well the end product performs, even though this is an inevitable consideration. Therefore, the Acceptance Criteria must in the end define the minimum basis for acceptance.
- Will assume the completion of integration testing. By the time acceptance testing will begin in the new system development effort, the bulk of testing will have already proved that the end product performs satisfactorily. This is done against an exhaustive set of trials that may be unreasonable to re-execute for the acceptance process.
- May enable acceptance by stages. The acceptance process may be facilitated by achieving acceptance in components or stages in accordance with the architectural limitations, avoiding a "big bang" approach. A staged process can also improve cash flow when payments are dependent on acceptance.
- Will be capable of objective demonstration within a limited time period. Acceptance Criteria will establish exactly what constitutes a successful demonstration. They will not be open-ended, allowing the user to ask for more and more proof of correctness.
- Will not simply restate requirements or design documents. Acceptance Criteria should be succinct. Do not rephrase other documents. Instead, reference other documents or deliverables, including those that have not yet been completed.

### 3.5.1 Indicative Acceptance Criteria for the Application Software

Acceptance Criteria for the software testing may be quite varied and extensive. One approach to planning Acceptance Criteria is to review them from several perspectives like:

- Internal program perspective. Possibilities include:
  - Compliance with coding standards;
  - Compliance with Naming Standards.
- Business flow perspective. Possibilities include:
  - Each process flow in the Business Process Model View has been implemented in software;
  - Individual screens and programs interact in the sequence described in the Process Flow Diagram. After process flow breaks, processes restart and produce the results anticipated;
  - Batch jobs run to completion;
  - Other business scenarios are met.
- External function perspective. Possibilities include:
  - Compliance with User Interface Standards for format, behavior, use of function keys, field labeling, use of colors;
  - Compliance with the Design Baseline established during the Development Phase.
- Recovery perspective. Possibilities include:
  - A specific number of transactions (representing a sample of online programs) can be aborted in mid-transaction and be shown to leave the databases intact;
  - Batch jobs can be restarted from every planned step-restart point, as described in the Operations Documentation;
  - A checkpoint/restart capability is provided for mainframe batch or background programs anticipated to run in excess of some time period, for example, 60 minutes. The results of each program of this type can be proven to be identical whether the program is restarted from a checkpoint or is run without interruption.
- Performance perspective (on-line). The system can be demonstrated to achieve a total throughput of X transactions per second for a period of Y minutes given a specific scenario described by:
  - Number of simulated or actual terminals;
  - Database of a specified size in number of rows or records;
  - Transaction mix;
  - Machine size and capacity;
  - Level of contention from other systems.
- Performance perspective (batch). The system can be demonstrated to complete a specified list of batch functions in less than X hours given a scenario described by:
  - Number of transactions (if any) to be posted in batch;
  - Size of database in rows or records;
  - Number of on-request reports specified;
  - Machine size and capacity;
  - Level of contention for system resources.

## 3.6 Testing and Acceptance Phases

Systems' changes and releases will be tested in five sequential phases: unit and link, integration, system (FAT), and preliminary site Acceptance (Pre-SAT) and site acceptance (SAT). Unit, integration and system testing will be conducted by the Technical team at its premises. The Customer will be responsible for pre-SAT and SAT testing phases. The Technical team will support the Customer during these two last acceptance phases. The five test phases are described below.

Phase 1 – Unit and Link Testing

The unit testing will involve testing of the individual software units or groups of related units. A unit is a component that is not subdivided into other components; it is a logically separable part of a computer program. Each unit of code will be evaluated on how well it meets the performance requirements for which it was designed. Timing, memory, accuracy in producing numerical and logical results will be considered as well as the preparation of input and output required for validating program logic, syntax, and performance requirements.

The unit testing conclusions should clearly state if the module under test is ready for integration with the rest of the modules, based on the number of bugs outstanding. After successful unit testing, link (integration) testing will be conducted to test the result of putting together the modules and sub-modules which successfully passed the unit testing.

Link tests will be designed to provide assurance that dependent groups or sequences of the software units (module clusters, program clusters, remote dialogue responders) can communicate in the way intended. This is not a test of functionality, although obviously some functionality will be tested in the process of testing linkage. Link testing is a valuable aid to demonstrating technical coexistence of software units, as a precursor to the more expensive system testing, where delay caused by a simple interface error can be embarrassing and costly.

The unit and link testing will be performed by the Technical team's team members responsible for writing the code. Activities throughout the unit and link testing will involve:

- Define and prepare the unit and link tests;
- Define and prepare the unit and link tests input data;
- Set up and configure the unit and link test environment;
- Run unit and link tests;
- Collect and process results;
- Prepare and issue the Unit and Link Test Report.

The primary output of the unit and link testing phase will be the Unit and Link Test Report which the testing session is described and the results are presented and analyzed.

*Phase 2 – Integration Testing*

After the successful completion of the unit and link testing phase, the integration phase will involve the integration of external systems with the new system and the verification that each new external system is integrated with the product and satisfies their assigned requirements.

A prerequisite of the integration testing is the qualification testing of the product that is to be integrated in the system in order to verify its basic performance.

Activities during this phase will include:

- Generate Integration Test Data - Building actual data into files, databases or test harnesses, using values stated in specifications.
- Conduct Integration Tests - Performing steps according to scripts, registering timings, printing diagnostics, and noting all significant events.
- Analyze Integration Test Results - Revisiting test outputs systematically, checking accuracy and consistency, and identifying non-conformity.
- Support Corrective Actions - Diagnosing and correcting faults, re-testing according to standards completing the relevant documentation under configuration management.
- Integration Test Report - Evaluation of the complete test process and production of the Integration Test File which will contain information on test log and results analysis, error correction and re-test and final reporting.

Each requirement identified in the Software Requirements Specification will be tested during the integration testing. This traceability ensures that the new system will satisfy all of the requirements and will not include inappropriate or extraneous functionality.

*Phase 3 – Factory Acceptance Testing (FAT)*

The Factory Acceptance Testing (FAT) or System Testing of the developed system represents the full business and functional testing of the system before the delivery to end users and concerns only the full (major) release of the system, and not eventual patches.

The FAT will verify that the new application meets its specifications and that all development activities are completed and finally that all the test scripts to be executed during the FAT phase have passed successfully. For the FAT session the following requirements must be met:

- The FAT environment is set up, stable and as close as possible to the (potential) production environment;
- The FAT environment is completely separate from the development and integration environments;
- The software and related data infrastructure and content (software package) installed in the FAT environment are the results of a successful (final) Integration Testing Session / Phase.

In case minor / cosmetic problems are detected during the FAT session, the software release under FAT could be accepted under certain conditions (the sort out of the problems detected with a pre-defined deadline). A blocking problem detected during the FAT Session could terminate the FAT Phase and send back to develop the application under FAT but the recommendation is to continue the FAT session as far as possible in order to check the remaining functionalities. The final decision to accept or not the developed software will be taken by the Project Manager based on the acceptance recommendations issued by the Technical team. The acceptance recommendations are concluded by comparing the results of the FAT Session to the FAT acceptance criteria (quality indicators) defined for the developed software.

- Activities during the FAT phase will involve:

- Generate System Test Data - Building actual data into files, databases or test harnesses, using values stated in specifications.
- Conduct System Tests - Performing steps according to scripts, registering timings, printing diagnostics, and noting all significant events.
- Analyze System Test Results - Revisiting test outputs systematically, checking accuracy and consistency, and identifying non-conformity.
- Support Corrective Actions - Diagnosing and correcting faults, re-testing according to standards completing the relevant documentation under configuration management.
- System Test (FAT) Report - Evaluation of the complete test process and production of the System Test File which will contain information on test log and results analysis, error correction and re-test and final reporting.

The output of the FAT session is the FAT or System Test Report which will contain a presentation of the FAT session (application to be tested, FAT environment, FAT data, FAT tests, acceptance criteria) and the corresponding results and conclusions. The FAT Report will clearly state if the current release of the software is accepted internally by the consortium and if it is ready to be delivered to the end users.

The Consortium's technical project team will run the FAT session and the Project Manager and/or the Quality Manager will supervise the FAT session and take the acceptance criteria. The Customer can be present at the FAT session as evidence on the successful completion of the FAT. After end users' accept the results of the system testing (based on the FAT Report), the Technical team will be allowed to deliver the new application for acceptance.

*Phase 4 – Preliminary Site Acceptance Testing (Pre-SAT)*

The Preliminary Site Acceptance Testing (Pre-SAT) Phase has the role of checking that the software release (officially) delivered by the technical team complies with a set of basic quality requirements. The set of basic quality requirements are project dependent but as a minimum include: the smooth (automatic) installation and configuration of the software release in the target environment (SW and HW) plus the bug free behavior of the main functions of the delivered application. The Pre-SAT Phase will be executed in a specific environment set up on the end user premises.

The Pre-SAT Phase will include but is not limited to the following activities:

- install and configure (if necessary) the software delivered as described in the accompanying release note;
- test basic, new and corrected functions of the system;
- collect and process results;
- issue a Go Ahead decision;

The only output of the Pre-SAT Phase will be the decision to pass the developed application to the SAT Phase, if Pre-SAT OK ("delivery acceptance") or on the contrary to reject the delivery ("delivery rejection"). A Pre-SAT report will be generated and made available to the interested parties.

*Phase 5 – Site Acceptance Testing (SAT)*

Site Acceptance Testing is conducted to determine whether the developed system satisfies its acceptance criteria and to enable the Customer to determine whether to accept the new system release. The acceptance test is required to validate that the developed system, its related documentation, tools, and hardware, satisfy all of the specified requirements and objectives of the end user, the requirements specification, and the design criteria. Acceptance testing will include tests of all intra-system interfaces; and the use of all manuals, documentation, procedures, and controls.

The consortium will be responsible for running the testing according to the acceptance test plan as agreed with Customer. The expected activities will involve:

- Prepare the test environment, generate acceptance test data and scripts etc;
- Support testers on the acceptance test site and/or remotely;
- Support corrective actions by diagnosing and correcting faults, re-testing according to standards completing the relevant documentation under configuration management.

The only result of the SAT Session is the SAT Report produced by the testers. This document will contain a description of the SAT environment (acceptance environment, application software specific release, data structure and content), the testing related issues (testing scenarios, cases, tests), the test results and SAT Session conclusions. If the SAT Session is successful, then the developed software delivery will be installed in production and an acceptance note will eventually be issued by the end user and sent to the consortium. If not, the Technical team will take the appropriate actions.

The end of each SAT will be marked by a technical meeting where the final decision to accept/reject with reservation/reject the developed system will be taken. In the latter case, the changes agreed in the meeting shall be implemented and a new Pre-SAT/SAT cycle will be planned.

Each test will deem to be successful if the actual test result matches exactly the expected result as defined in the specifications. If this is not the case, the end user will raise an issue and report that the test has failed.

In case that all tests have been completed satisfactorily, Customer will deliver a Certificate of Acceptance that will show the acceptance date and mention any reservations.

## 3.7 Test Levels

**Test Levels define the Types of Testing to be executed on the Application Under Test (AUT**). The Testing Levels primarily depends on the scope of the project, time and budget constraints.

In the project LOCARD, there are 5 types of testing that will be conducted.

- **Unit testing:** individually test each software module specified in LOCARD technical annex.
- **Infrastructure testing:** test infrastructure for deploying software modules, subsystems and the integrated platform.
- **Integration Testing** (Individual software modules are combined and tested as a group)

- **System Testing**: Conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements
- **API testing**: Test all the APIs create for the software under tested

## 3.8 Bug Triage

The goal of the triage is to

- To define the type of resolution for each bug
- To prioritize bugs and determine a schedule for all "To Be Fixed Bugs'.

## 3.9 Test Completeness

- Specifies the criteria that denote a **successfu**l completion of a test phase
- **Run** rate is mandatory to be 100% unless a clear reason is given.
- **Pass** rate is **90%**, achieving the pass rate is **mandatory**

# 4 Test Deliverables

The Test deliverables are provided as below:

### 4.1 Before testing phase

- Test plans document.
- Test cases documents
- Test Design specifications.

### 4.2 During the testing

- Test Tool Simulators.
- Test Data
- Test Trace-ability Matrix - Error logs and execution logs.

### 4.3 After the testing cycles is over

- Test Results/reports
- Defect Report
- Installation/ Test procedures guidelines
- Release notes

# 5 Collaborative tools for software design and development and testing

In the LOCARD project will be focusing on the collaboration between partners and on the code sharing. Namely, the following tools will be used to foster the code sharing, collaboration between partners, bug tracking and automatic integration

## 5.1 GitLab

GitHub is a Git-based repository hosting platform with millions of users world-wide. Similar to GitHub, **GitLab is** a repository manager which lets teams collaborate on code. Written in Ruby and Go, **GitLab** offers some similar features for issue tracking and project management as GitHub.
**Gitlab** is a service that provides remote access to Git repositories. In addition to hosting code, the services provide additional features designed to help manage the software development lifecycle.

## 5.2 Slack

**Slack** is essentially a chat room for your whole company, designed to replace email as your primary method of communication and sharing. Its workspaces allow you to organize communications by channels for group discussions and allows for private messages to share information, files, and more all in one place.
Plus, Slack integrates with a host of other apps so you can manage your entire workflow through one platform.

## 5.3 Jenkins

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purposes. **Jenkins is used** to build and test your software projects continuously making it easier for developers to integrate changes to the project and making it easier for users to obtain a fresh build. Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCM tools including CVS, Subversion (SVN), Git, Mercurial, Perforce, Clearcase and RTC, and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands.

Builds can be started by various means, including being triggered by commit in a version control system, scheduling via a cron-like mechanism, building when other builds have completed and by requesting a specific build URL. Continuous integration has many advantages:

- When unit tests fail or a bug emerges, developers might revert the codebase to a bug-free state, without wasting time debugging
- Developers detect and fix integration problems continuously — avoiding last-minute chaos at release dates (when everyone tries to check in their slightly incompatible versions)
- Early warning of broken/incompatible code
- Early warning of conflicting changes
- Immediate unit testing of all changes
- Constant availability of a "current" build for testing, demo, or release purposes
- Immediate feedback to developers on the quality, functionality, or system-wide impact of code they are writing
- Frequent code check-in pushes developers to create modular, less complex code

- Metrics generated from automated testing and CI (such as metrics for code coverage, code complexity, and features complete) focus developers on developing functional, quality code, and help develop momentum in a team

Every server running an instance of Jenkins offers an overview of the build status for all projects that were configured to be built on the server. This overview page is implemented with the help of an Apache HTTP server. For most users and administrators this is the most important way of getting up-to-date information about current builds and configuration of their Jenkins instance. A well-configured Jenkins server allows access to this page only after proper login, depending on access permissions administered by an LDAP server. While the configuration manager is allowed to trigger builds through this web page, the project manager will only get status information. This is in contrast to software developers who usually have access permission for more detailed build results like automated error analysis through static code analysis.

## 5.4 Jira

JIRA is a tool developed by Australian Company Atlassian. It is used for bug tracking, issue tracking, and project management. The name "**JIRA**" is actually inherited from the Japanese word "Gojira" which means "Godzilla". The basic use of this tool is to track issues and bugs related to your **software** and Mobile apps. It is also used for project management. The JIRA dashboard consists of many useful functions and features which make handling of issues easy.

JIRA is used to capture and organize the team's issues, prioritize and take action on what is important, and stay up-to-date with the project tasks. The team spends less time managing the work and more time building the software. The tool provides the team with the following functionalities:

**Process**
JIRA has workflows to match the team’s existing processes that can easily be adapted as the team evolves.
**Planning**
Teams that need to be productive and efficient choose JIRA to help them capture, assign, and prioritize their work. JIRA ensures that everyone on the team knows exactly what needs to be done and when, leading to a flawlessly completed endeavor.
**Collaboration**
On any team, it's important that people can easily share information and reach out for help when they need it. JIRA's simple, intuitive interface allows people to collaborate with other team members and get the job done more efficiently.
**Visibility**
JIRA allows the monitoring of issues that are most important to each team member, of activity streams, and facilitates the sharing of information with powerful dashboards, wallboards, and more.

## 5.5 Version Control System

As teams design, develop and deploy software, it is common for multiple versions of the same software to be deployed in different sites and for the software's developers to be working simultaneously on updates. Bugs or features of the software are often only present in certain versions (because of the resolution of some problems and the introduction of others as the program develops). Therefore, for the purposes of locating and fixing bugs, it is vitally important to be able to retrieve and run different versions of the software to determine which version(s) the problem occurs in. It may also be necessary to develop two

versions of the software concurrently (for instance, where one version has bugs fixed, but no new features (branch), while the other version is where new features are worked on (trunk).

At the simplest level, developers could simply retain multiple copies of the different versions of the program, and label them appropriately. This simple approach has been used on many large software projects and while this method can work, it is inefficient as many near-identical copies of the program have to be maintained. This requires a lot of self-discipline on the part of developers, and often leads to mistakes. Consequently, systems to automate some or all of the revision control process have been developed.
Moreover, in software development, legal and business practice and other environments, it has become increasingly common for a single document or snippet of code to be edited by a team, the members of which may be geographically dispersed and may pursue different and even contrary interests. Sophisticated revision control that tracks and accounts for ownership of changes to documents and code may be extremely helpful or even indispensable in such situations.

Revision control may also track changes to configuration files. This gives system administrators another way to easily track changes made and a way to roll back to earlier versions should the need arise.
Traditional revision control systems use a centralized model where all the revision control functions take place on a shared server. If two developers try to change the same file at the same time, without some method of managing access the developers may end up overwriting each other's work. Centralized revision control systems solve this problem in one of two different "source management models": file locking and version merging. **Based on the software development methodology that is applied in our team, the best preferred tool for version control of the source code is the version control provided by Git**.

### 5.6 Apache JMeter

The Apache JMeter application is an open source Apache project pure Java application designed to load test functional behaviour and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions. Apache JMeter can be used as a load testing tool for analysing and measuring the performance of a variety of services, with a focus on web applications. It can be used as a unit test tool for JDBC database connections, FTP, LDAP, Web Services, JMS, HTTP, generic TCP connections and OS Native processes. It can be used on both static and dynamic resources (Web Services (SOAP/REST), Web dynamic languages - PHP, Java, ASP.NET, Files, etc. -, Java Objects, Databases and Queries, FTP Servers and more). JMeter can also be configured as a monitor, as an ad hoc monitoring solution and can be used for some functional testing as well.

JMeter supports variable parameterization, assertions (response validation), per thread cookies, configuration variables and a variety of reports. The JMeter architecture is based on plugins, with most of its "out of the box" features implemented with plugins. Its usability can be easily extended with custom plugins during development. Its features include:

- Ability to load and performance test many different server/protocol types:
    - Web - HTTP, HTTPS
    - SOAP / REST
    - FTP
    - Database via JDBC
    - LDAP

- Message-oriented middleware (MOM) via JMS
- Mail - SMTP(S), POP3(S) and IMAP(S)
- MongoDB (NoSQL)
- Native commands or shell scripts
- TCP

- Complete portability and 100% Java purity.
- Full multithreading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- Careful GUI design allows faster Test Plan building and debugging.
- Caching and offline analysis/replaying of test results.
- Highly Extensible core:

  - Pluggable Samplers allow unlimited testing capabilities.
  - Several load statistics may be chosen with pluggable timers.
  - Data analysis and visualization plugins allow great extensibility as well as personalization.
  - Functions can be used to provide dynamic input to a test or provide data manipulation.
  - Scriptable Samplers (BeanShell, BSF-compatible languages and JSR223-compatible languages)

## 5.7 Swagger Tools

Swagger is an open-source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful web services. While most users identify Swagger by the Swagger UI tool, the Swagger toolset includes support for automated documentation, code generation, and test-case generation.
Swagger allows you to describe the structure of your APIs so that machines can read them. The ability of APIs to describe their own structure is the root of all awesomeness in Swagger. Why is it so great? Well, by reading your API's structure, we can automatically build beautiful and interactive API documentation. We can also automatically generate client libraries for your API in many languages and explore other possibilities like automated testing. Swagger does this by asking your API to return a YAML or JSON that contains a detailed description of your entire API. This file is essentially a resource listing of your API which adheres to OpenAPI Specification.
There are a lot of factors that contributed to Swagger's meteoric adoption for building RESTful APIs. We have listed the most important ones as to why use Swagger for designing APIs:

- Design and document APIs simultaneously, thus keeping the blueprint and documentation in sync.
- Both human and machine readable, with an interactive API documentation automatically generated to see the API in action.
- Large comprehensive tooling ecosystem around the framework, which lets you go beyond just design, from SDK generation to testing and debugging.
- Strong open source community supporting and spearheading the framework.

## 5.8 Auditing and the Monitoring Solution

Our proposed solution for the Auditing and Monitoring of the Extranet Platform, is based on the Nagios® Core software, an Open Source system and network monitoring application. It watches hosts and services

that are specified by the Administrator, alerting when problems arise and when they are solved. Some of the many features of Nagios Core include:

- Monitoring of network services (SMTP, POP3, HTTP, NNTP, PING, etc.)
- Monitoring of host resources (processor load, disk usage, etc.)
- Simple plugin design that allows users to easily develop their own service checks
- Parallelized service checks
- Ability to define network host hierarchy using "parent" hosts, allowing detection of and distinction between hosts that are down and those that are unreachable
- Contact notifications when service or host problems occur and get resolved (via email, pager, or user-defined method)
- Ability to define event handlers to be run during service or host events for proactive problem resolution
- Automatic log file rotation
- Support for implementing redundant monitoring hosts
- Optional web interface for viewing current network status, notification and problem history, log file, etc.

Nagios Core is licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation, thus providing legal permission to copy, distribute and/or modify Nagios under certain conditions. Read the 'LICENSE' file in the Nagios distribution or read the online version of the license for more details.

# 6 Infrastructure Environment

The development, testing and staging environment comprise of the hardware servers and operating systems that will be deployed for the whole course of the project. The specifications for the hardware servers - including operating systems - to cover all required work is as follows:

- 2VMs (under ESX)
- Each VM powered by 2 x Intel Xeon SP Gold 16-Core @ 2.10GHz
- 3.84 TB NVMe SSD Datacenter Edition
- 96 GB DDR4 ECC RAM
- Linux Ubuntu 18.x, Linux CentOS
- VMWare vSphere License
- VPN connections to all technical partners.

The servers are hosted on a secure datacenter that fulfills ISO 27001 security requirements.

# 7 Software Development Approach

This section presents the technical team's approach for the continuous improvement, design and development of the Web Resources based on agile/lean approaches as well as the continuous delivery of new features, improvements and modules.

## 7.1 Agile/lean development

The technical team will adopt the Scrum methodology in order to provide an agile value-driven approach to the delivery of the products. Scrum is an iterative, incremental methodology for the management of software development. It is a holistic approach that increases the speed and flexibility for projects with rapidly changing or at a large extent unknown requirement. This makes it suitable for the current project considering that new functionality, currently unknown, will be developed and that users provide a lot of feedback on a daily basis, which means that only a small initial set of user needs is known at the time of the tender. The key principles of the proposed methodology are:

- A self-organising and cross-functional team - Tasks allocation is decided by the team as a whole, covering the complete implementation lifecycle from idea to implementation;
- Splitting the work in a prioritised list of small and concrete deliverables accompanied by the amount of effort needed to complete each of them;
- Dividing time into short fixed-length iterations (2 weeks according to best practises) potentially producing software releases;
- Collaborating closely with the product owner (i.e. the Onsite Product Manager) and optimising the release plan, updating priorities of tasks to be performed after each iteration;
- Optimising the process by having a Sprint retrospective after each iteration;
- Facilitating the resolution of an increased volume of incoming emergency tasks by changing Sprint parameters (duration, prioritisation, etc.).

In addition to the Scrum principles outlined above, the technical team's approach will utilise principles of **Disciplined Agile Delivery (DAD)** to:

- address the full delivery life cycle from project initiation to production;
- add a **risk-driven viewpoint** to the value-driven approach; and
- add **lean governance** techniques to balance self-organisation.

The following sections describe the main elements of the technical team's methodology as adapted to the requirements of the project. They include the Roles, Events and Artefacts of the approach.

### 7.1.1 Roles

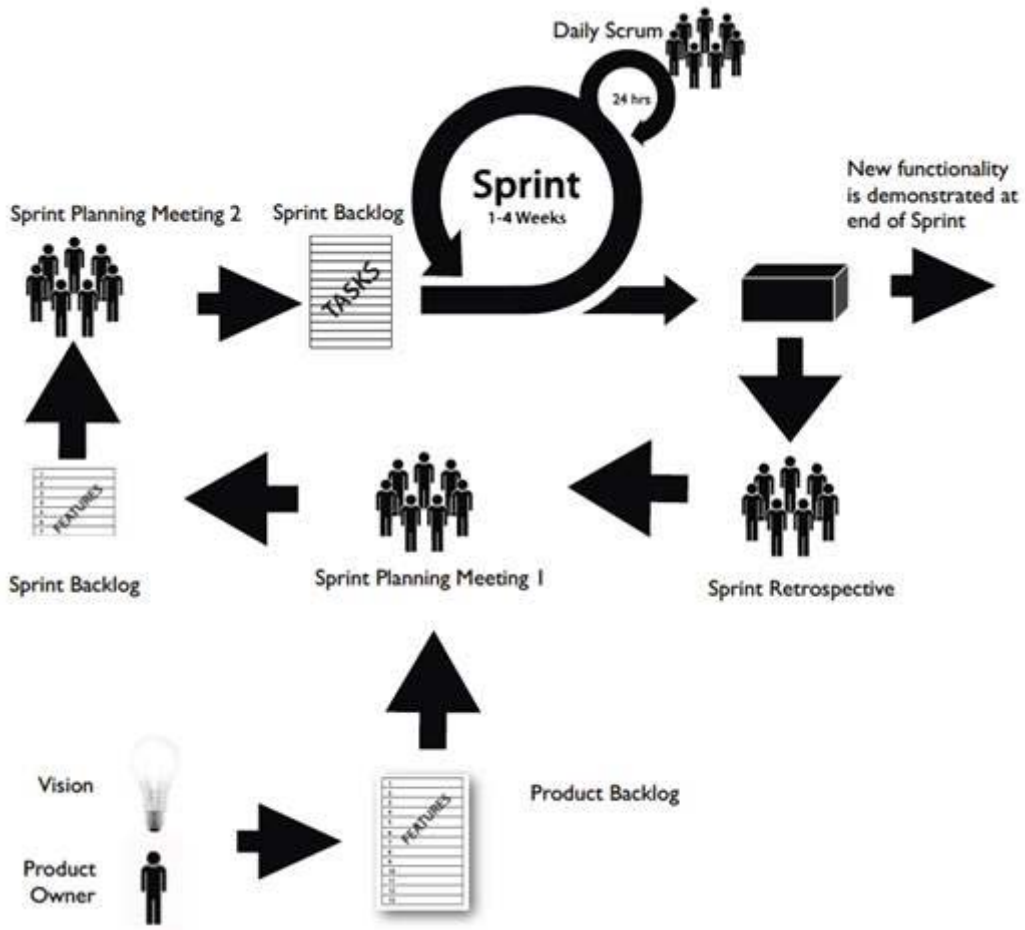
The key roles under the proposed approach are the following:

1. **Integrated Platform Manager**. The role will be taken by the leader partner of wp5 who is also managing the whole technical teamwork and effort. The person for this role will also be the liaison between the technical team and the end users as well as the other partners of the consortium.
2. **Stakeholders / End Users**. They include all stakeholders, i.e. end users, Client officials, Client Project Manager and more. A stakeholder is everyone potentially affected by the development and/or deployment of a software project/feature/etc.
3. **Scrum Master**. The Scrum Master is the agile coach who leads the team and keeps them focused on delivering work items and fulfilling their iteration goals and commitments that they have made. S/he is responsible for the day-to-day management and monitoring of the Scrum Team in the context of the Scrum framework proposed. The Scrum Master will be of the profile Project and Quality Manager (PM).
4. **Scrum Team Members**. The Scrum Team is a cross-functional group of individuals that will perform configuration, development, testing, analysis, architecture, design, planning, estimation and other activities as appropriate throughout the specific projects. The team focuses on producing the actual solution for stakeholders. The team will include the profiles of resources covering all the areas of expertise necessary for the successful implementation of each specific project. LOCARD's technical team has appointed several people to fulfill this role.
5. **Architecture Master**. The architecture master makes the architecture decisions for the team and facilitates the creation and evolution of the overall solution design. Architecture is a key source of project risk and someone needs to be responsible for ensuring the team mitigates this risk.

These roles stem from Disciplined Agile Delivery and the technical team's experience in similar contracts and expand the three basic roles of Scrum by including the Stakeholders and the Architecture Master.

### 7.1.2 Events

A Sprint (or iteration) - as presented in the following diagram - is the basic unit of product development in scrum. The Sprint is time-boxed effort. For LOCARD, the Sprint duration is initially defined to 3 weeks, which is flexible according to the needs of the specific project. A built-in buffer will also be taken into account to allow for uncertainty and change.




Each Sprint starts with a Sprint planning event, the aim of which is to define a Sprint backlog, where the work for the Sprint is identified and an estimated commitment for the Sprint goal is made. Each Sprint ends with a Sprint review and a Sprint retrospective where the progress is reviewed and shown to stakeholders and lessons-learnt for the next Sprints are identified.

### 7.1.3 Ad hoc Meetings

The technical team will have an open communication channel with end users during the whole duration of the project in order to ensure constant progress and alignment in the requirements elicitation, analysis, clarification, prioritisation, user feedback evaluation, etc. This means that several teleconferences or meetings may take place between the technical team and end users at any time during a Sprint execution. In these meetings, the technical team will also report on the project's progress in a similar, informal manner as in daily/weekly scrums. The main objective of these meetings is to speed up the implementation process by providing instant feedback and to keep the technical team and end users fully aligned.

### 7.1.4 Platform Backlog Refinement

Five to ten per cent of each Sprint is usually dedicated by the technical team to refining the Platform Backlog. This percentage may change according to the needs of the project. This includes regularly scheduled weekly meetings, covering:

- detailed requirements analysis follow up;
- splitting large items into smaller ones;
- estimation and revision for new items; and
- re-estimation of existing items.

This refinement activity will not be for items selected for the current Sprint; it will be for future (next one or two Sprints) items. With this practice, Sprint Planning will be relatively simple because the Platform Manager and Scrum Team start the planning with a clear, well-analysed and carefully estimated set of items.

### 7.1.5 Artefacts

#### 7.1.5.1 Platform Backlog

The work to be done during each Sprint, following the principles of Scrum, will be listed in the Platform Backlog, a refined and prioritised list of features that will be created by the technical team and maintained throughout the project life-cycle. Backlogs will be managed via electronic tools, where all team members will have access to and update.

The proposal backlog consists of features, non-functional requirements, bug fixes, security requirements, etc. It includes everything that needs to be done in order to successfully deliver a viable system, module, feature, etc. The platform backlog items are ordered based on considerations like risk, business value, dependencies, date needed, etc. Items added to a backlog are usually written in story format (user stories), however this is not mandatory. Platform Backlog items can be articulated in any way that is clear and sustainable, such as user stories, use cases, or any other requirements approach that the team finds useful.

#### 7.1.5.2 Sprint Backlog

The Sprint backlog is the list of work the scrum team must address during the next Sprint. The list is derived by selecting backlog items from the top of the modules/platform backlog until the scrum team feels it has enough work to fill the Sprint. The velocity of the team should be used as a guideline of how much work the team can complete. Tasks on the Sprint backlog are signed up for by the team members as needed according to the set priority and the scrum team member skills. This promotes self-organisation of the scrum team and developer buy-in. A task board is also used for the state of the tasks of the current Sprint (e.g. "to do", "in progress" and "done"). Once a Sprint has been delivered, the product backlog is analysed and re-prioritised, if necessary, and the next set of functionalities is selected for the next Sprint.

#### 7.1.5.3 Release Burn-down Chart

Scrum includes a Release Burn-down chart that shows progress towards the release date. The release burn-down chart is updated at the end of each Sprint by the scrum master. The release burn-down chart helps to drill down into a Sprint and understand what is the remaining work, what work has been added, what work has been done, what work will have to be done, as well as the work that the team has completed and how scope changed.

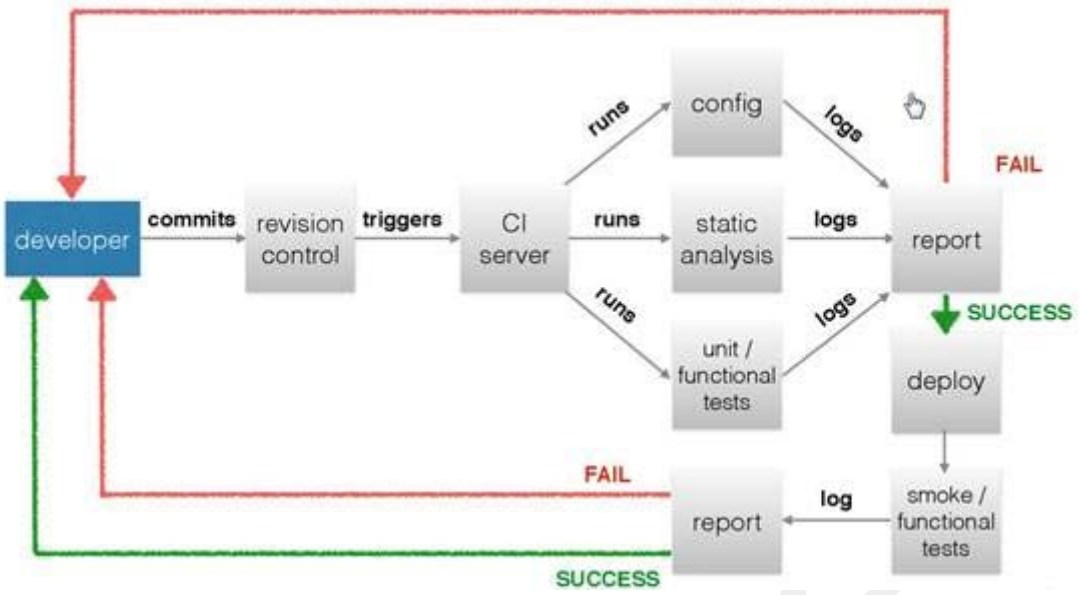
#### 7.1.5.4 Product Increment

The platform increment is the sum of all the platform modules backlog items completed during a Sprint and all previous Sprints. When a Sprint ends, the increment must be complete and in a usable condition regardless of whether the product owner decides to actually release it. The module / platform increments that are released are available to the users for testing until replaced by the next release.

## 7.2 Continuous Delivery

The technical team will use proven practices for the continuous integration and delivery of software products. Centralized repository code versioning, continuous integration/deployments and automated software testing and installation/configurations are just some of those practices. Continuous delivery builds on top of the agile software build process known as continuous integration. The process is managed by an automated code building tool (**Jenkins for LOCARD**), which provides clear user interfaces and a configuration console to manage software build, test and deployment. The deployment pipeline includes a set of validations through which a piece of software must pass on its way to release. Code is compiled if necessary and then packaged by a build server every time a change/configuration is committed to a source control repository, then tested by a number of different techniques (in some cases including manual testing) before it can be marked as releasable.

Another aspect of the continuous delivery process is presented in the workflow below, which includes the most common tasks of the process performed by a developer.




Management of all application server components will be done from a centralized place. Set of predefined configurations and actions are defined in a structural view, so they can be extended and executed in an easy way. Configurations are a set of property files, which can be configured on the fly depending on targeting environment, host or specific configuration. Actions are a set of scripted files, which are used often during management and administration and which replace manual intervention. All configurations and actions are saved under the central repository system and with automation of those, preparation and installation of already configured components can be repeated in a fully automated way. This means that if there is a need to create a new system or to extend some part, with setting only configuration, it can be prepared in the automated way within a short period of time, making it to be portable and easily extendable.

Maintaining and administration of all components is done solely from one place and each action can be monitored. This way the impact of miss-configuration and possible human errors during configuration is minimised, which will make the system compatible with Continuous Delivery principles.

# 8 Terms/Acronyms

| TERM/ACRONYM | DEFINITION |
|---|---|
| API | Application Program Interface |
| AUT | Application Under Test |
| *FAT* | *Factory Acceptance Testing* |

| | |
|---|---|
| *Pre-SAT* | *Preliminary Site Acceptance Testing* |
| *SAT* | *Site Acceptance Testing* |

# 9 Conclusion

In this document a thorough analysis of the testing methodology, test deliverables, testing tools, testing environment and collaboration tools is presented enabling seamless and continuous integration. The Agile methodology the technical team will follow is also presented. Moreover, all software tools that will be used for testing and collaboration are presented and the testing environment was described both in terms of software and hardware.
This deliverable is closely linked to work in wp3 and especially D3.3, D3.4 and D3.5 since what will be tested - including end user requirements, proposed system requirements and system architecture as well as LOCARD software modules - will follow the methodology presented in this deliverable. Furthermore, this deliverable is forming the basis and acts as input to wp6 and specifically task 6.1 testing and validation protocols. But wp6 will also provide feedback to wp5 in terms of several qualitative and quantitative metrics in an iteratively way and along platform integration cycles during the life of LOCARD.
D5.1 is horizontal for LOCARD project and will follow all modules and system design, software development, testing and final commissioning to end users with guidelines of all actions that have taken place during the testing period.

# 10 References

**Jira:** https://www.atlassian.com/software/jira
**Jenkins:** https://jenkins.io/
**Slack:** https://slack.com/intl/en-gr/
**GitLab:** https://about.gitlab.com/
**JMeter:** https://jmeter.apache.org
**Nagios:** https://www.nagios.org
**Swagger**: https://swagger.io/docs/specification/2-0/what-is-swagger/
**OWASP:** https://owasp.org/
**Penetration Testing checklist:**
https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Web_Application_Penetration_Checklist_v1_1.pdf
**Penetration Testing:** https://www.sans.org/reading-room/whitepapers/auditing/conducting-penetration-test-organization-67
**Agile methodology**: https://www.agilealliance.org/