17/02/2016

Deliverable D8.1
Service Validation and Testing Process

Deliverable D8.1

Contractual Date:	31-01-2016
Actual Date:	17-02-2016
Grant Agreement No.:	691567
Work Package/Activity:	SA4
Task Item:	T1
Nature of Deliverable:	R (Report)
Dissemination Level:	PU (Public)
Lead Partner:	PSNC
Document Code:	GN4-1-16-22F69

Authors: M. Wolski (PSNC), I. Golub (CARNet), G. Frankowski (PSNC), A. Radulovic (MREN), P. Berus (PSNC), S. Medard (RENATER), S. Kupinski (PSNC), T. Appel (DFN/LRZ), T. Nowak (PSNC), S. Visconti (GARR), I. Smud (CARNet), B. Mazar (CARNet), B. Marovic (AMRES), P. Prominski (PSNC)

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 691567 (GN4-1).

Abstract

The services introduced in NRENs production networks are composed of many individual components: people, hardware, software, environment, licenses, etc. The GN4-1 Activity SA4 defined the Service Validation and Testing Process in order to ensure only quality-tested products reach the production environment. D8.1 describes this comprehensive quality evaluation, which is an important part of the GÉANT service lifecycle.

Table of Contents

Table of Figures

Table of Tables

Executive Summary

SA4 performs a comprehensive quality evaluation in order to confirm that the examined service is ready for production. The foundations of the quality evaluation conforms to the ITIL Service Validation and Testing Process. The purpose of this process is to ensure that both customer expectations on a service (i.e. those who use the service) as well as IT operations requirements (i.e. those who will deliver, deploy, manage and operate the service) can be met. The ultimate result of the evaluation is a recommendation on service readiness for production, together with proposed improvements to the service development and maintenance. These constitute the key input to other processes, including the change management plan.

Essentially, the SA4 service validation and testing objective is to ensure that every service entering production is validated against a minimal set of criteria, determining if it can be reliably and efficiently operated. This quality evaluation engages an independent team of SA4 experts from the software, security and system administration domain. The SA4 team conducts all necessary tests – a software quality review, security code audit, system testing, and others – to evaluate the service quality based on the acceptance criteria. During the evaluation process, end users' representatives from NRENs are contacted to provide their feedback through a survey about the user support and general system usability.

Service assessment and testing is based on a structured and standardised approach to testing. This approach is aligned with the accepted standards in the field and complements ITIL recommendations and processes. It is also in agreement with the common language and practices in testing. The proposed schema can be thought of as a tool for combining the interests of software developers, testers and the computer networking community, and forms a framework for evaluating various products and solutions developed and applied within the GÉANT project.

1 Introduction

GÉANT services have been part of the NREN's production networks for a number of years. The services are composed of many individual components: people, hardware, software, environment, licenses, etc. Many of these components are just end-products, some are composed of even smaller individual parts and putting them together and deploying them into production to support NRENs' business processes can be a challenge.

GN4-1 Activity SA4 (Production Application Services and Infrastructure) corresponds to the need for a rigorous service validation and testing process before a service goes into production. Such validation has been recognised as an important part of the GÉANT Product Lifecycle Management (PLM) which should be performed in cooperation with, but independently from, the service development teams.

The purpose of the service validation and testing effort is to:

- Plan and implement a structured validation and test process that will provide evidence that the service will support its users' business within agreed service levels.
- Provide quality and security assurances for the release of the service, its components and service capability.
- Identify risks, issues, and errors as an input for the next phases of the service: transition to production.

The priority for SA4 is to deliver high quality and reliable services and the service validation and testing process in SA4 is a key element of the service transition. Prior to entering the production phase, every service must be validated against a lean, but necessary, set of criteria, to test the reliability of its operation. As a result, problematic issues should be identified and solutions applied before the service is launched in a production environment.

There are many activities within this process; all are logically structured and have three distinct phases: planning and preparation, testing and reporting.

The rest of the document is structured in the following way. Section 2 presents the service validation and testing process and introduces its activities, roles and responsibilities. Section 3 introduces the planning and preparation phase that prepares the assessment and plans its execution. Section 4 explains how the service assessment is realised through the software, documentation and service review, system testing and user support preparation. Section 5 describes how the results of the assessment are reported to form the input to other processes.

The report is extended with Appendixes presenting test documentation schema, User Support Survey and System Usability Scale survey. The document also contains references and a general glossary of testing-related concepts and acronyms used in the text.

2 The Service Validation and Testing Process

Service Validation and Testing (SVT) has been introduced in the GÉANT project as a separate task for the first time in GN4-1 within the SA4 activity (SA4 T1). The purpose of the task is in line with the service validation and testing process (SVT) defined in ITIL [ITIL], to ensure that a new or changed service or service offering is *fit for purpose* and *fit for use*. Fit for purpose determines what the user gets (the external view on a service from the perspective of a service provider). It is an indication how the service supports the target business performance or remove constrains [ITIL]. Thus the external quality reflects the conformance to business and user demands, which can be measured by overall user satisfaction.

The fitness for use determines how the service is delivered, which is an internal view on a service from the perspective of a service provider. The fitness for use reflects service capabilities like availability, reliability, capacity and security. The internal quality can be evaluated against a set of operational criteria, derived from the service design – management and operational requirements as well as the defined by the operation teams – best practices and standards. The internal view also includes software quality aspects. Software quality is basically understood as a lack of defects ("bugs") in the product, but it can be also determined by a set of software product metrics [Kan2002].

The SVT process leans on the GN4-1 Product Lifecycle Management [PLM] structure for services developed and provided through in the GÉANT environment and within GÉANT projects. The SVT process relies on the PLM to confirm that the service candidate has a valid prototype and that it has successfully passed the pilot phase. The prototype phase is a proof-of-concept to demonstrate a technology or to show the implementation of a selected use case. The pilot phase aims to validate the service conformance with user and business requirements as well as to define the product management strategy and assess market opportunities. Only when a service passes these two phases, it becomes a valid candidate to start the transition to production.

2.1 Process review

The service candidate is evaluated to confirm its capabilities to work in a production environment. In particular, the evaluation takes into account:

- IT operations view on a service i.e. those who deliver, deploy, manage and operate the service.
- Users expectations i.e. those who use the service.

Key actors involved in the SVT are: Service Development Activity (SDA) that hosts Service Delivery Team (SDT), Service Quality Team (SQT) and Service Operation Manager (SOM). The SDA is an entity responsible for the service development. The SDA initiates service transition to production, which in turn triggers the validation and testing process. The Service Operation Manager is one of the service stakeholders. His/her main role is to coordinate the SVT process execution for the service in order to ensure that all business and user requirements will be considered properly during the process execution. SQT comprises the technical staff, including system analysts, testers, system administrators, network engineers and others, that is formed independently from the SDA group to conduct quality assessment against the service stakeholders' requirements.

The SVT consists of the three main phases, as presented in Figure 2.1:

- Planning and preparation – includes gathering of necessary requirements from the development teams including service documentation, working instance and source code, gathering resources and parameters to define and complete various tests, risks management, SVT time-frame, etc.
- Service assessment – carrying out manual and/or automated testing in the area of service validation, secure and quality audits, functional and non-functional testing of service, code, techniques and procedures. Tests are planned to harmonise various aspects of a service and to be relevant and repeatable. All results are registered.
- Reporting – test results are documented, evaluated against the given criteria, gathered in the report which is then provided to the target stakeholders and other relevant processes including change management and continual service improvement.

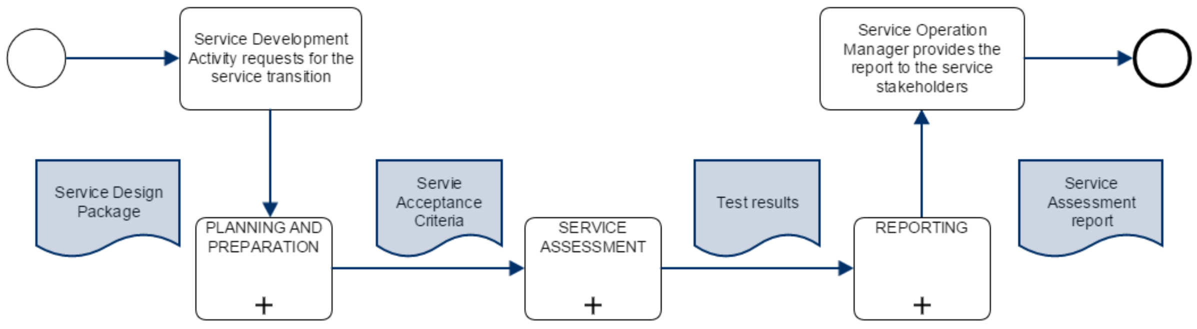
Service Development Activity requests for the service transition
Service Operation Manager provides the report to the service stakeholders
Service Design Package
PLANNING AND PREPARATION
Servie Acceptance Criteria
SERVICE ASSESSMENT
Test results
REPORTING
Service Assessment report

Figure 2.1 The SA4 service validation and testing process

The three main phases are described in detail in the sections 3 to 5.

3 Service Validation Planning and Preparation

A request for service validation is authorised at first by the PLM team and the service readiness to production is validated against the criteria defined by the PLM and/or SDT team through the SDP. When PLM confirms that the service candidate has successfully passed all previous service lifecycle phases, then SA4 enters in the first phase of the SVT process.

The main goal of this phase is to gather information needed for the SVT team in order to be able to assess the service. As the SVT team is organised independently from the Service Development Team (SDT), so there is a possibility that some SVT team members will face the service for the first time.

ITIL Service Design defines a useful set of documentation in the Service Design Package (SDP) [ITIL]; this needs to be passed from the SDT to SVT team. Service documentation may also contain links to additional service documents. SDT should also provide access to a stable operational instance of a service and the service source code.

The SDP includes also operational requirements, service Key Performance Indicators (KPIs) and metrics, as well as standards and best practices for service operation and management. All of these are reviewed by the Service Operations Manager (SOM), who may request additions and/or changes, in accordance with the operational teams.

All the inputs to SVT, including the delivered SDP, are reviewed by the appointed SOM. The outcome of this phase is the Service Acceptance Criteria (SAC) for the service assessment. The SAC comes originally from the SDP.

3.1 Acceptance criteria

The SAC contains a set of criteria used to ensure that a service meets its expected functionality and quality and that the service provider can deliver the new service once it has been deployed. The SAC reflects the service operational requirements and/or requirements of other actors involved in the service operation and management, such as the CSI manager or Service Desk function. The SAC is an interface between the SVT team and other actors involved in the service operation and management. For example:

- The SOM may request to examine operational procedures against specific metrics.
- The CSI team may request to assess service KPIs and their correlation on software metric.
- The service desk may request validation that the existing service resources can perform the first or second level of user support effectively.

Table 3.1 presents one such SAC example:

Criteria	Parameters	Test pass/fail criteria
Does the service ensure the given response time in case of a larger workload?	Response time (RT) Workload size (WS)	RT < 5 seconds when WS <= 100 request per second
Is the service architecture scalable and allow to extend the configuration to handle larger workload?	Workload size	Yes/No
Does the SDP contain the completed list of basic operational documentation?	Documentation checklist	Yes/No
Does the existing user support channels ensure effective communication with user communities?	N/A	Yes/No
Does the service deployment procedure handle the software stack update?	N/A	Yes/No

Table 3.1: The example Service Acceptance Criteria

The single SAC can be parametrised to determine the given requirements more precisely. The test pass or fail criteria can be provided for a SAC item as an input for the SVT process.

The SAC is essential for the SVT. It defines the scope of the SVT process as well as the threshold determining how different issues detected during SQA are classified (more about the classification can be found in the following section).

A future release of the SVT may have the SAC producing a formal contract between the SVT team and the SVT requestor.

4 Service Assessment and Testing

Assessment as well as testing is a cognitive, critical and often creative process. It builds upon experience, discipline and persistence, but also benefits from thinking out of the box. It benefits from the wise application of standards, tools, best practices, adapted to a specific case, goal, available resources, and local culture. Even though the SVT process includes a set of different tests, all of them share a common testing framework that is presented in the following section.

4.1 Testing Framework

The approach to testing design and specification has to be structured and aligned as much as possible with the major standards in the field (such as ISO/IEC/IEEE 29119, IEEE 829 and BS 7925), complemented with ITIL recommendations and processes, but is at the same time in agreement with the common language and practices of testing.

The testing is specified with the test plan. In order to produce the test plan, its creators need to familiarise themselves with the content and context in order to define the scope, organise its development, identify and estimate risks, establish an approach towards them, design a strategy for the particular testing and determine the staffing profile and schedule. The draft plan then needs agreement or approval, before distributing it to all involved.

The stakeholders monitor progress through the test status and completion reports and assess related measures and metrics. They issue directives for corrective and adaptive actions on the test design, environment and execution, which may result in changes in the test level documentation. Ultimately, these measures may lead to modification of the test plan.

The initial test specification documentation should be provided by the development team in the SDP set, but can be adapted and is finalised by the SVT team. It refines the practical details of the design and execution of the work specified in the test plan, taking into consideration traceability and repeatability, and captures relevant information gathered during testing in order to support the evaluation level reports.

Appendix A introduces a set of documents that could be provided as a result of testing preparation and execution.

4.2 Service Validation Tests

The Service Assessment and Testing (SAT) phase takes the SAC for the service evaluation as an input. The SVT team executes the necessary tests in order to evaluate the service quality based on the SAC. There are several test categories conducted within the SVT process:

- Software code audit – reviews the source codes quality and security. It includes an automatic and manual code review which can reveal potential bugs and problems, detect flaws in software design or discover source code security flaws. It is a white-box testing method i.e. it verifies the internal structure of the software.
- System testing – examines the service operational capabilities through the non-functional testing, service security testing through penetration and configuration testing, and usability testing to explore the usability factors and (optionally) service accessibility.
- Review of the basic service capabilities – this includes the quality of public software documentation and service architecture validation against different features (e.g. deployment flexibility/scalability or stability/availability/resilience). It is a black-box testing method, examining the service capabilities without investigating its internal structures or workings.
- User support evaluation – this aims to describe how the current user support is served and identifies possibilities for improvement.

The phase is completed with the acceptance status and any recommendations for improvements to the current development and maintenance process of the service.

The test scripts are recorded in the common repository [GITrepository] and can be re-used and repeated if needed, for the service in scope or for additional services.

4.2.1 Software code audit

Software code audits include a software quality review and a secure code review. Both audits include an automatic and a manual code review. Automatic code review relies on software tools that examine the source code of a solution and looks for potential bugs, duplicate code and patterns of bad code architecture.

Automatic code review is simple, fast and should be run on the whole source code. It may be integrated with software for continuous review. SonarQube [SonarQube] is an example of an open source platform for continuous inspection of code quality that covers several programming languages including Java, C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, and others.

Manual code review is a very important part of the audit, as it verifies the code from another perspective, in some cases unreachable with automatic tools. Manual audits mean that the service source is examined in detail by humans, and thus requires much more time, skill and resources to be performed then the automatic audits. While looking through source code and documentation, an expert may notice readability and/or design/architectural problems. However, an expert can only assess circa 150–200 lines of source code per hour, and may not be able to sustain that for the whole day, so reading the whole source code is suitable only for small projects. It is therefore a better approach to use automatic review results as an input for manual reviewing, and to use a manual audit as a complement or refinement of an automatic audit.

4.2.2 Software quality review

A software quality review starts with the automatic code review with (e.g. with SonarQube) and continues with the manual code review. The next step is to analyse results from an automatic code review. The person reviewing the code may decide to check every issue or only issues with higher significance. The expert should keep in mind the following points:

- Examine the code fragment related to an issue.
- Read the code fragment to evaluate if a reported problem is a true problem that needs a solution to increase code quality.
- Track the source of the problem and suggest at least one possible solution.

During the reviewing process, the reviewer should focus on:

- Readability.
  - Is it readable and adequately commented for readability?
  - Are the comments necessary, accurate and up to date?
  - Is the code over-commented or encumbered by trivial, formal, useless, or obsolete comments?
- Architecture/Design.
  - Classes/packages hierarchy.
  - Modules/Classes responsibilities.
  - Application layers.
  - Design patterns (Is the code as generalised and abstract as it could be? Should any feature be factored out into a separate class? Does code do too little or is it unnecessary? Is it reusable?).
  - Anti-patterns.
- Programming Rules.
  - Does it follow SOLID/OOP principles?
  - Does it have proper error handling?
  - Is it possible to improve its efficiency?
  - Is it possible to improve its security?

The output of manual code review is a document which lists the set of problems found in the project. Each problem is described by:

- The fragment where the problem was found (source code, documentation…).
- An explanation of the problem.
- A proposal for a solution.

An explanation alongside the relevant code fragment is the best way to achieve an understanding of the problem. Developers should apply the best advice to the project. Higher code quality would result in:

- Fewer bugs/errors and easier fixing issues.
- Easier development of new features or modification of already implemented parts.
- Easier code maintenance.

4.2.3 Security code audits

A security code audit is a clear box review – the auditor must have full access to the source code, and to a working service instance, together with all relevant documentation. The main goal is to find the largest possible number of source code security flaws and vulnerabilities. Information from the service documentation relevant to secure code audits are: the operating system, the programming language, the code size (number of lines), information on third-party modules in use including which of them are the most relevant parts from the service development team point-of-view. During the tests, the SDT may be asked to provide specific information about potential issues identified during the review.

The book "The Art of Software Security Assessment" [Dowd2006] presents three groups of strategies for secure code audit: Code comprehension-based, Candidate Points-based and Design Generalisation-based.

Because of the limited time and resources available for a code audit in SA4 T1, a mixed strategy is usually applied, and it includes two stages of the review:

- Interaction with the application in order to identify actual and potential bugs which then act as Candidate Points and are then verified on the source code level during the Stage 2. Additional candidates could be parts of the functionality that are recognised by the experts as critical (e.g. authentication, database communication, etc.).
- Actual source code review – confirmation of the found bugs in the source code, and additionally analysis of the suspicious parts of the source code, basing on the experience gained during the First Stage. For instance, if Stage 1 revealed several situations, where input data sanitisation is not sufficient, it means that all places in the source code where the applications accepts input data, should be inspected. The source code review should include an automated and a manual part – both are described below.

A secure code audit also starts with the automatic audit based on source code scanners, followed by the manual analysis in two areas:

- Verifying all results returned by the automated source code scanners (especially in terms of the false positives), because the tools may provide inaccurate results.
- Reading thoroughly the parts of the source code identified as Candidates (using preferably Code Comprehension – Analyse a Module/Class/Algorithm approach; for instance, reading the whole class responsible for communication with the database).

The main output of the audit is a detailed, technical report describing found vulnerabilities and providing recommendations (both allowing the removal or mitigation of the found vulnerabilities, and proposing general actions and recommendations for the future). If applicable or requested, an executive summary or presentation for the relevant persons may be prepared.

4.2.4 Penetration testing

Penetration testing is a thorough process of system security testing from a user's as well as inside and outside (black box) point-of-view, together with testing of the underlying operating system, other software package dependencies and its configurations. It is done by combining tools for conducting automated security tests together with manual checks. It is not a standard and structured process but it is dependent on the type of system being tested and its specifics, so it is not possible to define a testing scenario that can be used in all environments. Typical penetration tests include automated scans from external network without user login, and automated scans with user login.

Automated tools examine known generic issues but lack the ability to detect system-specific security flaws. They are also unable to detect new (zero day) vulnerabilities that may exist in the system under test. Results from automated tools require manual checks to be able to confirm the findings and to exclude any false positives. Manual checking of automated tools results is often a time consuming task, so an appropriate balance between the use of automated tests and manual checks is essential. Various testing guides are available and can be used for conducting manual security checks but the *de facto* standard in current web security assessment is the OWASP (Open Web Application Security Project) testing guide available at [OWASP].

The output of penetration testing is a report with information about the recognised system vulnerabilities. The next step will then be to correct the identified issues, depending on their severity.

4.2.5 Configuration testing

The main goal of the configuration review is to analyse the settings of one or more layers of the service environment and detect all inconsistencies with a selected template (e.g. a checklist) – including those that are very difficult or impossible to spot during penetration testing. The main input information is the list of the components to be analysed (e.g. CMS, Web server, database server, authentication server, operating system, Web Page language interpreter, etc.) and the configuration information for the agreed components. The information may be obtained in different ways, for example by granting the auditors access to the system under analysis, or by sending them the configuration files used by the Software Development Team or the environment administrator.

4.2.6 System testing – Service Documentation and Architecture Assessment

This assessment is basically a black-box test and first sanity check. Architecture assessment provides information of how the system works, explaining its main components, how data flows between components, what are the benefits of the chosen components and what are the potential risks. This information can also help identify any room for improvement and possibilities for optimising the system.

The output is the Quality of Service documentation, which includes:

- Software documentation (configuration, developer and user documentation).

- Service architecture (deployment/flexibility/scalability and stability/availability/resilience).

Quality of Service documentation means that:

- All relevant documentation is in place and available.
- Author details and the issue date of documents is known.
- Documents have a completed revision history.
- Private and confidential documents have appropriate access permissions.

Depending on the service, documents included in assessment have, for example:

- Software instance configuration parameters.
- Testing procedures.
- A production deployment guide.
- A production upgrade guide.
- Architectural overviews.
- Disaster recovery plans.

4.2.7 Non-Functional testing

The non-functional testing described in this section answer the question of "how" the system works, including the performance, operational, resilience, scalability, conformance, and usability tests. There are two main categories of non-functional requirements:

- Execution qualities, that focused on how the software behaves at runtime, such as usability or efficiency.
- Evolution qualities, such as scalability or maintainability, focused on the software structure.

Testing tools are selected according to service characteristics. In the non-functional testing of web applications, web browser add-ons like WebIDE, FireBug, Chrom DevTools are most useful. In web services API testing SoupUI is helpful. For all services performance tests, Apache JMeter, an open source desktop application, allows you to perform volume, stress and load tests by monitoring statistics such as response time and the system resources used.

Non-functional testing includes: operational, performance, resilience, scalability, conformance and usability testing, as described in the following sections.

4.2.8 Operational testing

Operational testing focuses on evaluation of all relevant service documentation and operational procedures. The main goal is to test documentation and procedures against:

- Completeness.
  - Are there any vital steps or explanations missing?
  - Are elements necessary for rebuilding and testing of the service available?
- Correctness.

    - Are there any errors?
    - Can all test cases be performed?
    - When the same procedure is executed many times are the results identical?
    - Are the screenshots accurate?
- Comprehensibleness.
    - Can a regular admin/user understand the instructions?
    - Is the content formatted for easy reading?

In order to verify the above three points, evaluators must apply all the steps from documentation just as the Operations Team would do, so the final result/state is identical to the expected outcome listed in documentation and carried out by the Operations Team.

4.2.9 Performance testing

The objective of performance testing is to measure how the system behaves in various predefined conditions. The required levels of performance are compared to the result of test execution [Lewis2008]. Performance tests could be combined from black-box and white-box tests, based on future services expectation in a production environment and the service architecture.

Performance test cases are usually based on functional test cases. The same scenario is performed under various loads of data or a simulation of a future load in a production environment, to check if the service meets the expectated KPI and to identify bottlenecks. The research performance problems have to spot inefficiencies in the code, at the database level (optimise queries), at the operating level (monitoring hardware resources), and eventually at the network level (analysing packets and protocols).

The acceptance criteria for performance tests should be from Service Operations and users and should refer to the non-functional requirements. For a web application, two criteria are more important: the expected load in terms of concurrent users (HTTP connections), and acceptable response times.

Testing software against performance requirements requires a clear methodology. Before performing any tests, a test environment should be established. A clone of a production machine of the service, with the same resources is also needed in order to tests the service limits. The tests are described in a plan that details all the different tests, defining the context needed to perform them, the input to provide, the actions to execute and the expected result. Three major statistics are monitored and collected by tools: the CPU, memory usage and the response time. These values help determine, according to acceptance criteria, if the test has succeeded or failed.

The perspective is to improve scalability and performance of the application. In performance testing, objectives can be defined that indicate different testing types:

- **Volume testing** – this tests how a system works with a large amount of data by filling the database or disk space of the application with lots of data and checking if the service still works efficiently. The service gathers data throughout its life, so this kind of test tends to check if the service will maintain good performance in this context.
- **Load testing** – investigates how the application behaves with a large but expected number of users. The goal is not to exceed the capacity of the software, but rather to place it in a context

of heavy use. To perform this category of test, the application database or disk storage area should be filled with a realistic amount of data before multiple concurrent users execute tests on the service. The purpose of these tests are to expose bugs that do not surface in functional testing with small amounts of data, such as a memory break. It helps to identify the maximum operating capacity as well as bottlenecks, and eventually to determine which element might cause a degradation of the service performance.

- **Stress tests** – tries to break the system under test by overwhelming its resources or by taking resources away from it (in which case it is sometimes called negative testing). The main purpose is to make sure that the system can handle an overload without crashing, but fails and recovers gracefully – this quality is known as recoverability. This form of testing is used to determine the stability of an application. For a web application, stress can be applied by greatly increasing the number of concurrent users or by running processes that consume large amounts of resource (Network, CPU, memory, disk) while using the application.

4.2.10 Resilience testing

"Resilience is the persistence of performability when facing changes". [Meyer2009] Resilience from the SA4 Service validation perspective is the ability of the service to provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation. The aim of resilience testing is to validate how a service recovers from crashes, for example when a service component is unreachable or a network connection to an external service is broken. It also covers a test of internal data consistency after an unexpected restart of services (for example after an operating system restart). The scope of a resilience test is based on the expectation given by the future Service Operation Team and assumed SLA levels. Test cases are based on an analysis of service architecture, internal component communication and type of communication with external services. Tests are the forced failure of the services in variety of ways to prove that the system work according to expectation after removing the network connection or the connection to an external database. The outcome from a resilience test can improve software error handing before production.

4.2.11 Scalability testing

Scalability testing is the testing of a software application for measuring its capability to scale up or scale out. This class of tests is used to determine if the user experience is affected by a significant data load. In the SA4 Service validation context, scalability testing requires an assessment of different parameters, as with the maximum number of simultaneous users accessing to a web page. The methodology consists in creating incremental loads of data and importing them in the software with the goal of testing if the system can cope with a high load. If the addition of resources allows the application to be equally or more efficient, the system is declared scalable.

Outcome from the scalability test is a survey of the resources needed by the system to prevent it from crashing or deteriorating in performance during its life.

4.2.12 Conformance Testing

Conformance testing is when services have to meet a rigorous protocol or standard and verifies that the service will work according to standard specification. For example, for services with http interfaces it should be checked that each HTTP response code is used correctly.

4.2.13 Usability testing

Usability in the international standard ISO 9241-11 is defined as: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

In the SA4 service validation framework there is a focus on examining usability factors and also on service accessibility [W3CAT]. Overall usability is covered by the System Usability Scale [Brooke1996] and provides a “quick and dirty” reliable tool for measuring the usability. It consists of a 10 item questionnaire with five response options on a Likert scale (from ‘Strongly agree’ to ‘Strongly disagree’). SUS gives global view of subjective assessments of usability. User responses could also be used to analyse each aspect, divided into the ability of users to complete tasks using the system or the level of resource consumed in performing various tasks in services by users. Accessibility in those test process is understood by key requirements for each kind of service. It is focused on providing that each service function is accessible to users and it is performed by expert testing.

Outcome of usability testing could be a set of references on how to improve user productivity and satisfaction e.g. improving visibility of the main services function on web pages.

4.2.14 User Support evaluation

User Support describe how the current support for the service is served to the end users. User support should be accurately defined as a process that captures, tracks, assigns, and manages incidents reported by the end user. ITIL defines an ‘incident’ as any event which is not part of the standard operation of a service and which causes, or may cause, an interruption to or a reduction in, the quality of that service. The objective is to restore normal operations as quickly as possible with the least possible impact on either the service or the user, in a cost-effective way [ITIL].

User support must be precisely defined and organised so that every level has well defined responsibilities in resolving the problems of end users. In the GÉANT project, there are three levels of service support: First Level Support – provided by the Service Desk in SA6; Second Level Support – provided by the SA4 T2; and Third Level Support – provided by the development team.

Evaluation of the user support is in two parts:

- Evaluation of the user support resources such as user guides and other documentation.
- Evaluation of the provided user supporting service.

Evaluation of the User support resources

End-user resources should include at least a Service user guide, FAQs, and multiple communication channels for at least First-level of support. These resources could be consolidated into a single User Guide document for user support.

The User Guide should be technical document intended to assist users in operating a service. It should provide a user-centred overview of the service and its use through the use of screenshots as illustrations of a service’s user interface, features, use scenarios, or the way the information is presented. Based on CBP documentation [SDBestPracticesGuide], all user guides should contain :

- Preface – the part containing the information on how to navigate through the document. It often provides basic information about the service and describes related documents.
- Table of Contents – provides a clear map of the location of the information provided in the document.
- Prerequisites section – inform the Guide's readers about software and hardware which user must fulfil for proper use of given service and guide. In most cases, this section contains descriptions of server/target machine hardware specifications, additional software which must be installed on the target machine with detailed instructions on how to do this, and an initial configuration. In more complicated cases, instead of detailed description of all necessary prerequisites this section may refer to the Hardware requirements specification and Software requirements specification documents.
- Typical Use Cases section – This section is a guide on how to use the core functions of the system.
- Troubleshooting section – This section provides information on how to deal with common issues while installing or operating the software. It provides information about known errors and workarounds.
- FAQ (Frequently Asked Questions) section – The FAQ section provides answers for the questions most frequently asked by users.
- Contact Details – Contact Details should provide comprehensive information about points of contact regarding technical support, feature requests, error reports, etc.
- Glossary – describes any acronyms or uncommon terms used in the documentation.

The User Guide should address one type of audience in a given user guide at a time (in some instances, providing specialised user guides for IT managers, project managers, application administrators, users, developers, etc.). The User Guide should be distributed with the service.

Good user support must be precisely defined and organised, this applies to resources also. The type of resource offered with the service and for whom the service is intended should be clearly defined. If resources are prepared well for the service, there will be less need to report problems and fewer calls for First level support in resolving them. The levels of support resources are as follows:

- First level: User guides for end users of service, FAQs, Wiki, Forums, all communication channels for the First level of support.
- Second level: User guides for developers and administrators, Wiki for developers and administrators, Forum for developers and administrators, all communication channels for the Second level of support.
- Third level: All communication channels for the Third level of support (i.e. third-party suppliers).

Evaluation of user support

Evaluation of user support may be carried out through a brief survey. The survey needs to determine how satisfied end users are with the user support they have received through all phases of using the service, from installation to use. The survey should also identify necessary improvements in the current user support. An example survey can be found in Appendix C.

There should be the widest possible participation in the survey. Only if a sufficient number of users, at least 50%, take a part can the results be used as a future recommendation for improvement of user support. When the survey is finished, results and feedback and recommendations for improvement are distributed to the operation and development team.

4.3 Issues classification

All the issues discovered during the SVT are marked with different levels of severity. Virtually all common classification examples use four levels, although the terminology may vary [QABestPractices]. Furthermore, an additional level can be used to describe any discrepancies that do not match the AC.

The reference issues classification is as follows [QABestPractices]:

- **Critical** – the problem has an extreme impact on the system, effectively blocking its usage.
- **Major** – the defect is severely restricting system usability.
- **Medium** – the defect does not prevent important functions of the system to be used.
- **Minor** – the defect does not interfere with the use of the system and user operations.
- **Not relevant** – not actually a defect, but there is some possible impact on the quality or perception of the system.

Both the issue's severity and the results of the analysis of its technical aspects are inputs for the service change management and release planning. While the issue severity determines the issue classification against the approved SAC, the priority is associated with solution scheduling. This approach allows to manage properly the responsibilities between quality evaluation, which is taken by the system analyst independently, and the decision in which order appropriate solutions should be applied, which takes into account the assessed feasibility, costs, business impact etc.

Examples for severity levels for secure and configuration testing are provided in Appendix B.

The combination of severity and priority defines the decision matrix – a key input for the change management plan leading the service to deploy in production environment:

	High Priority issues	Low Priority issues
High severity	Must be addressed immediately and before a system goes into production. It may for example be a security vulnerability that can lead to a whole system being compromised, or a blocking error that affects the main system function.	Must be addressed but does not stop a system being deployed into production. It may for example be a security threat making potential attacks easier or a major drop in system performance that is likely to appear in the production environment.
Low severity	Should be addressed in the next system release. It may for example be a recommendation for system usability improvements.	Could be address in future system releases. It may for example be a recommendation for better examples in the administration guide.

Table 4.1: The decision matrix for the classification of issues

4.4 Testing Repetition Cycle

Within the ITIL framework, the repetition of testing should be the result of other “Service Transition” processes like the “Change Evaluation” process and “Release and Deployment Management” process.

Most of the tests described are valid only for the analysis snapshot of the observed application/source code at the time of testing. Because of the inevitable further development of operating system and development platforms, some of the tests may need to be repeated and/or performed periodically, especially security-related tests.

Tests should be repeated if the number of found issues is high or if some recommendations required comprehensive modifications of the source code, and it should be performed as soon as the recommendations have been implemented. The retests should be focused on verifying whether the particular issues have been addressed and checking that the recommendations have not introduced any negative side effects.

Periodic tests may need to be performed if it is known that the application is still under development (e.g. a new version will appear or a new module will be added) or significant changes have been made to the code and/or the solution platform. These tests should be concentrated on the new functionality (together with its interaction with the old functionality) and be complemented with a general review of the whole source code in the context of potential new classes of vulnerabilities (e.g. the presence of recently deprecated functions).

5 Service Validation and Testing Reports

The concluding part of the Service Validation and Testing process includes completing the reports produced in individual tests and producing one or more final reports as the final part of the process.

Each of the tests described in the previous section will provide its own results with either recognised issues of different severity and priority, or with the report that no issues were recognised and reported.

In addition, the Testing Documentation in Appendix A defines several reporting types: Test Status Report, Test Completion Report, Test Data Report, Test Environment Report, Test Execution Log, Detailed Test Results and Test Incident Report. These may need replicating for different types of stakeholders: the Service development team, the Service testing team and the Service operational team.

All reports will be assembled into one or more documents and made available to the target readers' group, as well as to the Service Validation and Testing team, for further reference. For example, the test report for service development activity should use recommendations for improvements as the key input to the change management plan. The test report can be extended or updated with additional information, according to the change evaluation process.

Every identified issue, incident, problem, error and risk should be recorded and assessed with grades of severity and priority assigned, and addressed. In the final report, each issue should be described with a recommendation to solve the problem or to improve the service. As a follow-up step, the issues can be recorded in the issue tracking system by the team that will work on issue resolution, together with an agreed timeline.

Used test plans, scripts and data should be stored in relevant repositories (for example, JMeter test plan sources should be stored in the GÉANT Code Repository, Git or SVN), so that certain components can be reused.

Appendix A Test Documentation

The documentation described below provides a hierarchy of artefacts that is aligned with common industry practices and is generally compatible with the IEEE 829 standard and its sequel/update ISO/IEC/IEEE 29119-3. The documentation provides detailed templates for both traditional (sequential and iterative) and agile approaches.

Metadata

All documents should start with the following common elements (metadata):

- Organisational numeric identifier of the document – it may be omitted if the document name is used as the identifier.
- Descriptive and identifying name of the document.
- Name of testing (sub-)project or use case, if not clear from document name.
- Document date.
- Version number.
- Author or authors and their contact information.
- Version history (table with version numbers, dates, contributors and change descriptions).

Version histories, date(s) and authors are not needed if the master documents are kept up-to-date in a document-managed environment, such as a CMS system or Wiki. However, the main or corresponding author and document date need to be visible in self-contained standalone snapshots that are published on the web or shared by email.

References/Supporting Documents/Literature

The list of documents with their identifiers, names, version numbers and hyperlinks to individual documents should be provided.

At least the reference to the project plan should be present in subordinate documents. There is no need to reference other documents that represent the common background already listed in the project plan. Only the references that are crucial for understanding need to be provided in lower level documents, as well as those that point to the relevant external documentation that is not already referenced in higher-level documents.

A.1 Test Plan

The test plan outlines the operational aspects of execution of the test strategy for the particular testing effort. It provides an overview of what the system needs to meet in order to satisfy its intended use, the scope of the intended testing effort, and how the validation is to be conducted. The plan outlines the objectives, scope, approach, resources (including people, equipment, facilities and tools) and the methodologies and schedule of the testing effort. It usually describes the team composition, training needs, entry and exit criteria, risks, contingencies, test cycle details, quality expectations, tracking and reporting processes. The test plan may span several test suites, but it does not detail individual test cases.

A vital document, the test plan is an instrument of mutual understanding between the testing and development teams and management. In case of major changes or impediments, it should be updated as necessary and communicated to all concerned. Such updates may lead to further changes in documents that specify the test design, test cases, the data and environmental requirements. Given the comprehensive nature of the test plan, it should not over-specify any implementation details that are covered in subordinate test level documents: this is to avoid unnecessary back-propagation of changes.

The recommended structure of the test plan is as follows.

Metadata

The descriptive name should briefly, in three to six words, state which system is tested, the target aspects, features of components, and the level, type or purpose of the testing being conducted.

It should be followed immediately by a separate description of the testing level (unit, integration, system and acceptance) and/or type or subtype (functional, non-functional, alpha, beta, performance, load, stress, usability, security, conformance, compatibility, resilience, scalability, volume, regression, etc.).

References/Supporting Documents

All documents that support the test plan should be listed. They may include:

- Project plan.
- Product plan.
- Related test plans.
- Requirements specifications.
- High level design document.
- Detailed design document.
- Development and testing standards.
- Methodology guidelines and examples.
- Organisational standards and guidelines.
- Source code, documentation, user guides, implementation records.

Glossary

- Key terms and acronyms used in the document, target domain and testing are described here. The glossary facilitates communication and helps in avoiding confusion.

Introduction

- This is the executive summary part of the plan which summarises its purpose, level, scope, effort, costs, timing, relation to other activities and deadlines, expected effects and collateral benefits or drawbacks. This section should be brief and to the point.

Features to be Tested

- The purpose of the section is to list individual features, their significance and risks from the user perspective. It is a listing of what is to be tested from the users' viewpoint in terms of what the system does. The individual features may be operations, scenarios, and functionalities that are to be tested across all or within individual tested sub-systems. Features may be rated according to their importance or risk.

- An additional list of features that will not be tested may be included, along with the reasons. For example, it may be explained that a feature will not be available or completely implemented at the time of testing. This may prevent possible misunderstandings and waste of effort in tracking the defects that are not related to the plan.
- Together with the list of test items, this section describes the scope of testing.

Test Items

- This is the description, from the technical point of view, of the items to be tested, as hardware, software and their combinations. Version numbers and configuration requirements may be included where needed, as well as a delivery schedule for critical items.
- This section should be aligned with the level of the test plan, so it may itemise applications or functional areas, or systems, components, units, modules or builds.
- For some items, critical areas or associated risks may be highlighted, such as those related to origin, history, recent changes, novelty, complexity, known problems, documentation inadequacies, failures, complaints or change requests. Probably there had been some general concerns and issues that triggered the testing process, such as a history of defects, poor performance, changes in the team, etc., that could be directly associated with some specific items. Other concerns that may need to be mentioned can be related to safety, importance and impact on users or clients, regulatory requirements, etc. The key concern may be general misalignment of the system with the intended purpose, or vague, inadequately captured or misunderstood requirements.
- The items that should not be tested may be also listed.

Approach

This section describes the strategy that is appropriate for the plan level and in agreement with other related plans. It may extend the background and contextual information provided in the introduction.

Rules and processes that should be described include:

- Detailed and prioritised objectives.
- Scope (if not fully defined by lists of items and features).
- Tools that will be used.
- Needs for specialised training (for testing, tools used or the system).
- Metrics to be collected and granularity of their collection.
- How the results will be evaluated.
- Resources and assets to be used, such as people, hardware, software, and facilities.
- Amounts of different types of testing at all included levels.
- Other assumptions, requirements and constrains.
- Overall organisation and schedule of the internal processes, phases, activities and deliverables.
- Internal and external communication and organisation of meetings.

- The number and kinds of test environment configurations (for example, for different testing levels/types).
- Configuration management for the tested system, tools used and the test environment.
- Change management.

For example, the objectives may be to determine whether the delivered functionalities work in the usage or user scenarios or use cases, whether all functionalities required the work are present, whether all predefined requirements are met, or even whether the requirements are adequate.

Besides testing tools that interact with the tested system, other tools may be need, like those used to match and track scenarios, requirements, test cases, test results, issues and acceptance criteria. They may be manually maintained documents and tables, or specialised testing support tools.

Some assumptions and requirements must be satisfied before testing. Any special requirements or constrains of the testing in terms of the testing process, environment, features or components need to be noted. They may include special hardware, supporting software, test data to be provided, or restrictions in use of the system during testing.

Testing can be organised as periodic or continuous until all pass criteria are met, with identified issues being passed to the development team. This requires defining the approach to modification of test items, in terms of regression testing.

The discussion of change management should define how to manage the changes of the testing process that may be caused by the feedback from the actual testing or by external factors. This includes the handling of the consequences of defects that affect further testing, dealing with requirements or elements that cannot be tested, and dealing with parts of testing process that may be recognised as useless or impractical.

Some elements of the approach are further detailed in subsequent sections.

Item (and Phase) Criteria

This section describes the process and overall standards for evaluating the test results. It is not a detailed criteria for passing an individual item, feature or requirement.

The final decisions may be made by a dedicated evaluation team comprised of various stakeholders and representatives of testers and developers. The team evaluates and discusses the data from the testing process to make a pass/fail decision that takes into account the benefits, utility, detected problems, their impact and risks.

The exit criteria for testing are also defined, and take into account the achieved level of completion of tests, the number and severity of defects sufficient for the early abandonment of testing, or code coverage. Some exit criteria may be bound to a specific critical functionality, component or test case. The evaluation team may also decide to end testing on the basis of available functionality, detected or cleared defects, produced or updated documentation and reports, or the progress of testing.

If testing is organised into phases or parallel or sequential activities, the transitions between them may be controlled by corresponding exit/entry criteria.

If the testing runs out of time or resources before the completion or is ended by stakeholders or the evaluation team, the conclusions about the quality of the system may be rather limited, and this may be an indication of the quality of the testing itself.

Suspension Criteria and Resumption Requirements

These criteria are used to determine *in advance* whether the testing should be suspended or ended before the plan has been completely executed. The criteria are also used to determine when the testing can be resumed after the problems that caused the suspension have been resolved.

Reasons for the suspension may include the failure of the test item (for example, a software build) to work properly due to critical defects that seriously prevent or limit the progress of testing. Non-critical defects may also accumulate to the point where the continuation of testing has no value. Other reasons include a change of requirements from the client, system or environment downtime, or an inability to provide some critical component or resource at the time indicated in the project schedule.

Issues noticed during testing may have been noted before the start of testing. If there are too many of these, particularly if it is obvious that the system or the item cannot satisfy the pass criteria, it may be sensible to suspend testing. A smoke test may be required before full testing is resumed.

Deliverables

This section describes what is produced by the testing process. In addition to the test documentation that are described here, deliverable items may also include the test data used during testing, test scripts, code for execution of tests in testing frameworks and outputs from test tools. The deliverables may be the subject of quality assessment before their final approval or acceptance.

Activities/Tasks

This section outlines the testing activities and tasks, dependencies and estimates their duration and required resources.

Staffing and Training Needs

This is the specification of staff profiles and skills needed to deliver the plan. Depending on the profile of the personnel, it should also detail training requirements on the tested system, elements of the test environment and test tools.

Responsibilities

This section specifies personal responsibilities for approvals, processes, activities and deliverables described by the plan. It may also detail responsibilities in development and modification of the elements of the test plan.

Schedule

The schedule of phases should be detailed to the level that is reasonable given the information available at the time of planning. It should give the timing of individual testing phases, milestones, activities and deliverables and be based on realistic and valid estimates, particularly as the testing is often interwoven with the development. Testing is the most likely victim of slippage in the upstream activities, so it is a good idea to tie all test dates directly to the completion dates of their related developments. This section should also define when the test status reports are to be produced (continuously, periodically, or on demand).

Risks and Contingencies

This section, which complements "Suspension Criteria and Resumption Requirements", defines all risk events, their likelihood, impact and the counter measures to overcome them. Some risks may be testing related manifestations of the overall project risks. The exemplary risks are the lack of or loss of personnel at the beginning or during testing, the unavailability or late delivery of required hardware, software, data or tools, delays in training, or changes to the original requirements or designs.

The risks can be itemised using the usual format of the risk register, with attributes such as:

- Category.
- Risk name.
- Responsible tracker.
- Associated phase/process/activity.
- Likelihood (low/medium/high).
- Impact (low/medium/high).
- Mitigation strategy (avoid/reduce/accept/share or transfer).
- Response action.
- Person responsible for taking action.
- Response time.

The approach for dealing with schedule slippages should be described in responses to associated risks. Possible actions include the simplification or reduction of non-crucial activities, a relaxation of the scope or coverage, the elimination of some test cases, the engagement of additional resources, or an extension to the testing period.

A.2 Test Status Report

The Test Status Report is a one-time interim summary of the results of testing activities. It may describe the status of all testing activities or be limited to a single test suite. This report, as well as the test summary report, summarises the information obtained during test execution and recorded in test logs and incident reports. It must be informative and concise and should not concern itself with minor operational details.

Depending on the approach defined in the test plan, it may be produced periodically, on completion of milestones or phases, or on demand. If periodic, it may be a base for continuous tracking through progress charts. Individual status reports are also sources for the test completion report.

The report should start with metadata in the condensed form, but without version history, since the document itself is considered to be a one-time snapshot.

Summary

The document can start with totals of passed, failed and pending test cases, scenarios or tests, and identified defects (if individually tracked). It may also show the coverage of test cases or code, consumption of resources and other established progress metrics, in numbers and, if possible, charts. Close to this summary, a comprehensive assessment should be provided.

The details of the progress are expressed in the form of a table describing the outcome of execution of individual test cases or scenarios, cumulatively since the start of testing. Typical attributes are:

- Test case ID.
- Test case name.
- Date and time of the last execution.
- The last execution status (not run, failed, partial, passed) or counts of failed and passed test executions.
- Number of associated defects.
- Brief comment.

If there are more than a dozen items, consider grouping and sub-totalling the table rows according to the test suites, test items or scenarios (as listed in the project plan), test types or areas. If a single report addresses several suites, each should have a separate test status report, or at least its own totals and details tables.

Observations and Highlights

If needed, the document provides evaluations and recommendations based on the interim results and incidents encountered since the previous status report. It may also use red flags to draw attention. It should report the resolution of issues that were highlighted in the previous status report. The summary of activities conducted since the previous status report is optional. If present, is should exist in all status reports.

A.3 Test Completion Report

The Test Completion or summary report is a management report that brings together the key information uncovered by the tests carried out. It recapitulates the results of the testing activities and indicates whether the tested system has met the acceptance criteria defined in the project plan. This is the key document in deciding whether the quality of the system and the performed testing are sufficient for the decision that follows the testing. Although the completion report provides a working assessment of success or failure of the system under test, the final decision is made by the evaluation team.

This document reports all relevant information about the testing, including an assessment about how well the testing has been done, the number of incidents raised and outstanding events. It must describe all deviations from the original test plan, their justifications and impacts. The data provided should be sufficient for the assessment of the quality of the testing effort.

The narrative provided should be more detailed than in the test status reports.

Summary

It recapitulates the evaluation of the test items. Although the test completion report can reflect the structure of the test status report, the details that were only temporarily significant can be omitted. The table rows or subsections should correspond to the test items or scenarios listed in the test design specification. The summary should indicate the versions of the items that were tested, as well as the testing environment used. For each item it should be briefly explained what was tested and what was the outcome.

Test Assessment

This is a comprehensive assessment of the conducted testing. It should also indicate the areas that may require further investigation and testing.

Variances

All variations and discrepancies from the original test plan should be noted here. This section can also provide an assessment of differences between the test environment and the operational environment and their effect on the test results.

Test Results

This is a comprehensive interpretation of the test results. It includes a description of issues or defects discovered during the testing. It should also describe any unexpected results or problems that occurred during the testing. For resolved incidents, their resolutions should be summarised. For unresolved test incidents, an approach to their resolution should be proposed.

Evaluation and Recommendations

Propose decisions regarding the tested system and suggest further actions on the basis of the acceptance criteria, quality of the test process, test results and outcomes for individual test items. Provide recommendations for improvement of the system or future testing.

Activities

The summary of activities conducted during testing should record what testing was done and how long it took. In order to improve future test planning and save time it should be in an easily accessible form.

A.4 Test Design Specification

Test design specification addresses the test objectives by refining the features to be tested, the testing approach, test cases, procedures and pass criteria. This document also establishes groups of related test cases.

Features to Be Tested

This section describes the features or combinations of features that are the subject of testing. Each feature is elaborated through its characteristics and attributes, with references to the original documentation where the feature is detailed. The requirement descriptors include ID/short name, type, description and risks. The references lead to the associated requirements or feature specifications in the system/item requirement specification or design description (in the original development documentation). If the test design specification covers several levels of testing, the associated levels or types are noted for each individual requirement, feature or test item.

The features may be grouped into a few key applications, use cases or scenarios. If such groupings are made, a single feature or requirement may be present in several groups.

The use case is a high level description of a specific system usage, or set of system behaviours or functionalities. It should not be mistaken for a UML use case. It implies, from the end-user perspective, a set of tests that need to be conducted in order to consider the system as operational for particular use. It therefore usually describes the system usage, features and related requirements that are necessary for utilisation of the system by end users on regular basis.

The requirement is a description of a necessary capability, feature, functionality, characteristic or constraint that the system must meet or be able to perform. It identifies a necessary quality of a system for it to have value and utility to a user, customer, organisation, or other stakeholder. It is necessary for the fulfilment of one or several use cases or usage scenarios (in scenario testing).

The high-level requirements include business, architectural and stakeholder/user requirements. There are also some transitional requirements that are only relevant during the implementation of the system. The detailed requirements are defined on the basis of high-level features or requirements. Some of them are consequences of system's functions, services and operational constraints, while others pertain to the application domain.

Functional requirement defines a specific behaviour or function. It describes what the system does, but not how it does it, in terms of implementation, quality, or performance.

Non-functional requirements specifies the quality criteria used to assess the characteristics or properties the system *should* possess. Typical non-functional requirements include:

- Performance, availability, stability, load capacity, efficiency, effectiveness, scalability, response time.
- Reliability, robustness, fault tolerance, recoverability, resilience.
- Privacy, security, safety.
- Configurability, supportability, operability, maintainability, modifiability, extensibility.
- Testability, compliance, certification.
- Usability, accessibility, localisation, internationalisation, documentation.
- Compatibility, interoperability, portability, deployability, reusability.

In the sequential design process or waterfall model of software engineering, requirements are inputs into the design stages of development. The requirements specification is an explicit set of requirements to be met by the system, and therefore is usually produced quite early in its development. However, when iterative or agile methods of software development are used, the system requirements are iteratively developed in parallel with design and implementation.

The requirements specification is an important input into the testing process, as it lays out all requirements that should have been addressed during system development, so the tests to be performed could link back to them. Without access to the requirements from the development, the requirements that are directly associated with testing should be formulated during its planning. If an agile methodology is used for development, these requirements can reflect the completed Scrum epics, user stories and product backlog features or "done" Kanban board user stories and feature cards.

The individual requirements need to be consistent with the external documentation, verifiable, and traceable towards high-level requirements or stakeholder needs, but also towards the test cases. The requirements form the basis for the development of test cases.

Scenario testing is a higher-level approach to testing of complex systems that is not based on test cases, but on working through realistic and complex stories reflecting user activities. These stories may consist of one or several user stories, which capture what the user does or should do as a part of his/her job function, expressed through one or more sentences in the everyday or domain language. The tester who follows the scenario must interpret the results and judge whether they are considered a pass or failure, although this interpretation may require backing by domain experts. This term should be distinguished from the test procedure and the test case scenario.

Approach Refinements

This section refines the approach described in the test plan. The details of the included test levels are provided and how the individual features are addressed at those levels.

Specific test techniques are selected and justified. Particular test management, configuration management and incident management tools may be mandated. Code reviews, static and dynamic code analysis or unit testing tools may support the testing work. Test automation software tools may be used to generate, prepare and inject data, set up test preconditions, control the execution of tests, and capture outputs.

The method for the inspection and analysis of test results should be also identified. The evaluation can be done based on visual inspection of behaviours and outputs, or on the usage of instruments, monitors, assertions, log scanners, pattern matching programs, output comparators, or coverage measurement and performance testing tools.

In order to avoid redundancy, common information related to several test cases or procedures is provided. It may include details of the test environment or environmental needs, system setup and recovery or reset, and dependencies between the test cases.

If the test design includes some deviations from the test plan, they should to be described here.

Test Cases

Individual test cases are identified here. After the identifier, a brief description of the test case and associated test procedure is provided. Associated test levels or other important test case attributes may also be recorded.

The test case refines criteria that need to be met in order to consider some system feature, set of features or use case as working. It is the smallest unit of testing and is sometimes colloquially referred to as a ‘Test’. A single test case may be included into several test suites or related to a requirement associated with several use cases. If different test levels have separate test design specifications, a single test case may be present in several design specifications.

The selection of the test cases may be the result of an analysis that provides a rationale for a particular battery of test cases associated with a single requirement. For example, the same feature may be tested with distinct test cases that cover valid and invalid inputs and subsequent successful or negative outcomes. This distinction is made in terms of system responses and not testing outcomes, as reporting of an error may actually indicate that a test has actually passed. The logic behind the selection of test cases should be described here.

A feature from the test design specification may be tested in more than one test case, and a test case may test more than one feature. The test cases should cover all features, that is, each feature should be tested at least once. The relationship between the requirements/features and test cases is summarised in the Requirements/Test Cases Traceability Matrix, which is usually placed in a separate document that is updated with the evolution of the requirements and Test Design Specification. It enables both forward and backward traceability, as it simplifies how the test cases need to be modified with a change of requirements, and *vice versa*. It is used to verify whether all the requirements have corresponding test cases, and to identify for which requirement(s) a particular test case has been written for. The Requirements/Test Cases Traceability Matrix is a table where requirements and test cases are paired, thus ensuring their mutual association and coverage. Since there are always more test cases than requirements, the requirements are placed in columns, and tests cases in rows. The requirements are identified by their IDs or short names and can be grouped by type, while the test cases can be grouped into sections according to levels: unit, integration, system and acceptance.

Feature Pass/Fail Criteria

This specifies the criteria to be used to determine whether the feature or a group of features has passed or failed, on the basis of results of individual test cases.

A.5 Test Case Specification

The test case specifications are produced after the test design specification is prepared. The test case specification is a detailed elaboration of a test case identified in the test design specification and includes a description of the functionality to be tested and the preparation required to ensure that the test can be conducted. A single test case is sometimes associated with several requirements: it may be partially or fully automated.

The test case specification is provided within the use case or test-suite specification, a document that details all test cases, or even a separate document dedicated to a single test case. A formal written test case is characterised by a known preconditions, input and expected output and post-conditions, which are worked out before the execution.

For a system without pre-existing formal requirements, the test cases can be written based on the system's desired or usual operation, or operation of similar systems. In this case, they may be a result of decomposition of a high-level scenario, which is a story or setting description used to explain the system and its operation to the tester.

Alternatively, test cases may be omitted and replaced with scenario testing, which substitutes a sequence or group of test cases.

The typical attributes or segments of the test case specification are:

- Test case ID or short identifying name.
- Related requirement(s).
- Requirement type(s).
- Test level.
- Author.
- Test case description.
- Test bed(s) to be used (if there are several).
- Environment information.
- Preconditions, prerequisites, states or initial persistent data.
- Inputs (test data).
- Execution procedure or scenario.
- Expected post-conditions or system states.
- Expected outputs.
- Evaluation parameters/criteria.
- Relationship with other use cases.
- Whether the test can be or has been automated.
- Other remarks.

The test case typically comprises of several steps that are necessary to assess the tested functionality. The explained steps should include all necessary actions, including those assumed to be a part of common knowledge.

The test suite is a collection of test cases that are related to the same testing work in terms of goals and associated testing process. There may be several test suites for a particular system, each one grouping together many test cases based on a shared goal and functionality or shared preconditions, system configuration, associated common actions, execution sequence or reporting requirements. An individual test suite may validate whether the system complies with the desired set of behaviours or fulfils the envisioned purpose or associated use cases, or be associated with different phases of the system lifecycle, such as identification of regressions, build verification, or validation of individual components. A test case can be included into several test suites. If test cases descriptions are organised along test suites, the overlapping cases should be documented within their primary test suites and referenced elsewhere.

The test procedure defines the sequence of steps to be followed while executing a group of test cases (such as a test suite) or a single test case. It can explain the test setup, perform execution, evaluate results and restore the environment. The test procedures are developed on the basis of the test design specification and in parallel or as part of the test case specifications. Having a formalised test procedure is very helpful when a diverse set of people is involved in performing the same tests at different times and situations, as this supports consistency of test execution and result evaluation. The test procedure can combine test cases when the test cases are run in a fixed, logical order.

The test script is a sequence for instructions that is carried out in order to execute a test case, or test a part of system functionality. These instructions may be given in the form suitable for manual testing or, in automated testing, as short programs written in a scripting or general purpose programming language. For software systems or applications, there are test tools and frameworks that allow continuous or repeatable execution of prepared automated tests to be specified.

A.6 Test Data Report

This document describes the data used in testing. It is typically produced in two stages.

First, the data requirements implied by the test plan and test design are put together. They include requirements relating to type, range, representativeness, quality, amount, validity, consistency and coherency of test data. There may be additional concerns related to the sharing of test data with the development team or even end users.

If the test data are not entirely fabricated, but extracted from an existing database or service and can be associated with real services, business entities or persons, the policies and technical procedures for its anonymisation, or protection may need to be established. If data are generated, then the approach for the creation of adequate data should be established, and the validity of tests results obtained with such data elaborated. Test case specifications provide more details about the data needed and should be sufficient to support the actual collection or generation of adequate test data.

During the second stage, the selected tools are used to prepare these data for execution for all use cases, including their injection into the system. The expected test outputs are defined at this time, and, if possible, automated methods for comparing the baseline test data against actual results are devised. The limitations of test data and supporting tools are identified and mitigations are explored. Finally, the measures that ensure the usability and relevance of test data throughout the testing process need to be conducted. This includes data maintenance, data versioning and backup. The decisions and knowledge produced during preparation of the test data are captured.

A.7 Test Environment Report

This document describes the test environment. It is typically produced in two stages.

The requirements for the test bed implied by the test plan, test design specification and individual test cases are put together, and the initial test environment setup is designed. The test bed requirement related to the test level, system features and requirements, test items, test data, testing scenarios and procedures, chosen support, measurement and monitoring tools are also assembled. Security, safety and regulatory concerns are also considered. Policies and arrangements for sharing of the test bed and allocated resources with other teams or users are established.

The initial test bed design can be a simple deployment diagram or test bed implementation project, but it should cover all elements of the setup, including hardware, software, network topology and configuration of hardware, external equipment, system software, other required software, test tools, the system under test and individual test items. If some components are not immediately available, a staged implementation schedule or workarounds need to be devised. A walkthrough through at least the most important requirements and test cases needs to be performed in order to validate the proposed design.

The report is updated after the test environment is set up. A smoke test can be performed. Any limitations of the test environment are identified and mitigations devised. The maintenance plans, responsibilities and arrangements are established. If envisioned in the test plan, this may include upgrades of current versions of test items, resources, network topology, and other details of the configuration. The initial design is updated to reflect the test environment as it was built. The decisions and knowledge produced during the implementation of the test bed are captured.

A.8 Test Execution Log

The test execution log is the record of test cases executions and results obtained, in the order of their running. Along with test incident reports, the test execution log is the basis for test status and completion reports. These documents allow direct checking of the progress of the testing and provide valuable information for solving the cause of an incident.

This log provides a chronological record about the execution of tests, by recording which tests cases were run, who ran them, in what order, and whether the test was passed or failed. Tests pass if the actual and expected results are identical; they fail if there is a discrepancy. If the test design specification permit a "partial" pass, it must also clarify how to treat such outcomes within the feature pass/fail criteria and acceptance criteria.

Each test execution should start with a standardised header, with executions ordered from the oldest to the newest. Optionally, they may be grouped by test cases, but, if this is done, it is important to maintain the ability to trace the actual execution sequence of all test cases to detect possible interference between them. For each test execution, the versions of the system under test, its components, test environment, and specifics of input data must be recorded. The recommended descriptors are:

- Test case ID or short identifying name – may be placed at the top of the group.
- Order of execution number – useful for cross-referencing.
- Date and time.
- Testers – the people who run the test, may also include observers.
- Test bed/facility – if more than one test bed is used in testing.
- Environment information – versions of test items, configuration(s) or other specifics.
- Specific presets – initial states or persistent data, if any.
- Specific inputs – inputs/test parameters or data that are varied across executions, if any.
- Specific results – outputs, post-conditions or final states, if different from the expected.
- Execution status – passed, failed, partial (if permitted).
- Incident reports – if one or several test incident reports are associated with this execution.
- Comments – notes about significant test procedure steps, impressions, suspicions, and other observations, if any.

If the test execution log is natively maintained or presented in a human-readable format, and there are repeated executions of the same test case with same attribute values (tester, test bed, environment, presets, input, ...), then these common values can be documented in the heading of the group, along with the test case ID: this reduces clutter. However, any variation in the configuration, input data or results should be documented.

In the case of a deviation or partial success or failure, a more detailed description of the execution status should be provided.

The data captured in the test execution log, along with other test specifications and reports, should be sufficient to reproduce individual tests, that is, to recreate the needed setup, execute the test and produce the same or similar results.

If the testing is organised around scenarios instead of test cases, the general structure of the log is unchanged, except that inputs, states and outputs are replaced with interpretation of the interactions and results for each segment of the scenario, supported by key excerpts or snapshots of characteristic inputs and outputs.

A.9 Detailed Test Results

Detailed test results are the actual outputs, assertions, and system and monitoring logs produced during the execution of tests. They should be paired with the corresponding test execution log records. Their format may depend on the test tools used to capture them. In addition, the detailed results may encompass the reports produced by test automation tools that compare the baseline test data against actual results and which highlight deviations. Such reports are valuable traces of how actual results compare to expected post-conditions, states and outputs, and can be used to assess the execution status and write the test execution log and test incident reports.

A.10 Test Incident Report

The test incident report is used to document any event that occurs during the testing process that requires investigation. A discrepancy between expected and actual results can occur because the expected results are wrong, the test was incorrectly run, the requirements were inconsistent or unclear, or there was a fault or defect in the system or a problem with the test environment. It should provide all details of the incident such as the actual and expected results, when it failed, and any supporting evidence that will help in its resolution. All other related activities, observations, and deviations from the standard test procedure should be included, as they may also help to identify and correct the cause of the incident. Where possible, the report also includes an assessment of the impact of an incident upon testing.

The test incident report needs to be a standalone document, as it provides items of information that are already recorded in the corresponding test case and test execution log record.

A failed test may raise more than one incident, while an incident may occur in more than one test failure. The testers should try to identify unique incidents and associate them with the tested features or originating test items. This will provide a good indication of the quality of the system and its components, and allow any improvement to be monitored.

If an incident is the consequence of a fault or bug, the triggering error may be not in the failed execution, but in the previous one. It the case of apparently random incidents, the earlier executions and incidents should be checked in an attempt to recognise a pattern in the tests that led to them.

Metadata

The test incident report ID allows the report to be referenced in the test execution log and issue tracking system. If one or several test incident reports are raised or updated during a single test execution, they need to be recorded in the test execution log.

Summary

This briefly recapitulates the incident.

Description

The following elements should be recorded:

- Test case ID or short identifying name*.
- Order of execution number*.
- Date and time.
- Testers.
- Associated requirement/feature/test items.
- Test procedure step – where the event occurred*.
- Test bed/facility.
- Environment information.
- Presets*.
- Inputs*.
- Expected results*.
- Actual results*.
- Anomalies – discrepancies, errors or faults that occurred.
- Attempts to repeat – whether they were made, how, and what was the outcome.

Test case related details (marked with *) will be omitted if the incident in not linked to a specific test case or scenario. In such situations, a detailed narrative description should be provided.

Impact

If known, indicate the impact of the incident on test plans, test design specifications, test case specifications or test procedures.

Appendix B Security Vulnerability Severity Levels

Severity levels are adapted from the generally described approach. Functional errors receive "Not relevant" mark. Examples of security vulnerability levels are provided in Table B.1.

Severity	Description
Critical	A security vulnerability that directly allows a service or server to be compromised or stored data to be stolen. Examples: • Application code that apparently leads to an SQL Injection, easy to be identified and exploited (e.g. direct copying of a parameter from the URL to an SQL query, accessible without authentication in the service). • Lack of input data validation mechanism for a parameter which is passed to a function invoking a system call (like exec/popen/system in PHP).
Major	A serious security vulnerability but not leading to a direct compromise of service, server or data (it may be also). A gross deviation from security best practices. Examples: • Lack of input data validation mechanism for incoming data allowing to exploit a Cross-Site-Scripting vulnerability, while the session token is not protected with *httponly* security attribute. • Writing a cleartext password, entered by the user during login, to a logfile of the application.
Medium	Security vulnerability that does not directly lead to the application being compromised (it may require highly particular conditions or skills for that to happen), but it may deliver important information for an attacker or may be used as an auxiliary step in a sophisticated attack scenario. There may be significant deviation from security best practices. Examples: • Improper error handling, that leads to revealing internal structure of application – e.g. attaching the SQL server error message with the database tables and columns name to the error string displayed to the user. • Lack of input data validation mechanism for incoming data allowing to exploit a Cross-Site-Scripting vulnerability, while the session token is protected with *httponly* security attribute. • Insufficient commenting of security-relevant functions with non-intuitive source code.
Minor	Security vulnerability that reveals a small amount of relatively insensitive information. A small deviation from security best practices. A place where security hardening practices could be introduced. Examples:

Severity	Description
	• Instructions that cause throwing exceptions with revealing internal paths of the application location. • Creating password hashes using the SHA1 algorithm. • Constructing an HTTP response (Web page) without additional security headers like e.g. X-XSS-Protection. • Improper comment of a security-relevant function (e.g. copied and pasted from another function and erroneously left intact).
Not relevant	An error which is not associated with security vulnerability, either functional or in the process of software development. Examples: • A typo in the error message displayed to the user. • An instruction that causes displaying an improper page/window/message to the user. • A security irrelevant public class member that could be made protected or public. • Improper comment (or a lack of comment) in a security irrelevant part of the source code. • Unnecessary source code (e.g. a function that is never invoked).

Table B.1: Severity levels for Configuration testing – general description

Severity levels are adapted from the generally described approach. If the auditors consider that issues do not impact security, but may make using the system more difficult, they may describe issues as "Not relevant" to security. Examples are provided in Table B.2.

Severity	Description
Critical	A security vulnerability that directly allows a service or server to be compromised or stored data to be stolen. Examples: • An Apache Tomcat users.xml configuration file with data that allows to login to the management panel with default admin credentials. • PHP register-globals parameter set to *on* in a sensitive application. • Misconfigured SSL/TLS allowing SSLv2 or NULL ciphers. • An extremely old version of the application prone to publicly known security vulnerability with CVSS at least 7.0 with a public exploit (we recommend treating the application version number as a part of its configuration).
Major	Configuration setting that creates a security vulnerability not leading to a direct compromise of service, server or data (it may be possible, but very difficult) or revealing interesting data. A gross deviation from security best practices. Examples: • Misconfigured SSL/TLS: configured LOW ciphers, configured anonymous Diffie-Hellman key exchange. • Unconfigured HTTPS in an application that requires authentication, provided that the application does not work as root and does not invoke OS commands.

Severity	Description
	• A significantly old version of the application prone to publicly known security vulnerability with CVSS at least 7.0, but without a public exploit.
Medium	Security vulnerability that does not directly lead to the application being compromised (it may require highly particular conditions or skills for that to happen), but it may deliver important information for an attacker or may be used as an auxiliary step in a sophisticated attack scenario. There may be significant deviation from security best practices. Examples: • Apache mod_userdir enabled and user enumeration possible. • Misconfigured SSL/TLS: SSLv3 flag switched on or using MD5 algorithm as a digest function. • PHP: *expose_php* parameter switched on and the version revealed in HTTP server headres; PHP version is significantly outdated and there are known, but not critical, vulnerabilities known for that version. • Using moderately outdated software not affected by highly critical security vulnerabilities.
Minor	Security vulnerability that reveals a small amount of relatively insensitive information. A small deviation from security best practices. A place where security hardening practices could be introduced. A guideline for further similar installations, if it is probable that functional requirements enforce a setting not optimal from a security point of view. Examples: • Configuring Apache to reveal the detailed version numbers in HTTP Server header. • Configuring a Web server to handle excess HTTP methods like TRACE, OPTIONS, DEBUG. • Misconfigured SSL/TLS: no support for TLS 1.2 configured. • Activating PHP Easter Egg via *expose_php* parameter (PHP version is up-to-date or slightly outdated). • Using a little bit outdated software, prone to a limited amount of non-critical bugs (CVSS <4.0 or <7.0 and without known exploit).
Not relevant	Issue not associated with security, but detected during the review as a side-effect, written down for convenience. Examples: • Default session identifier name. • PHP: A security irrelevant function used in the source code but put into the *disable-functions* list.

Table B.2: Severity levels for Configuration testing – examples

Appendix C User Support Survey

This survey is part of SA4 T1's (Service Validation and Testing) task to determine whether the product, in this case 'NAME of SERVICE', is ready to proceed to the next phase of production.

One of the steps in this process of testing and validation of NAME of SERVICE is capturing the customer's point of view about this service. The main goal of this survey is to identify if there is scope for improvement in the user support for the current NAME of SERVICE.

1. On scale from 1 (strongly disagree) to 5 (strongly agree), please evaluate each statement about how are you satisfied with the way that **NAME of SERVICE** team provide you support during implementation of service

   - We are very satisfied with the quality of the support that **NAME of SERVICE** team provide us during implementation of service.

     1 – Strongly Disagree

     2 – Somewhat Disagree

     3 – Neither Agree nor Disagree

     4 – Somewhat Agree

     5 – Strongly Agree
   - We are very satisfied how fast **NAME of SERVICE** team resolved our queries during implementation of service.

     1 – Strongly Disagree

     2 – Somewhat Disagree

     3 – Neither Agree nor Disagree

     4 – Somewhat Agree

     5 – Strongly Agree
   - We are very satisfied with **NAME of SERVICE** team knowledge during implementation of service.

     1 – Strongly Disagree

     2 – Somewhat Disagree

     3 – Neither Agree nor Disagree

     4 – Somewhat Agree

     5 – Strongly Agree
   - We are very satisfied with the kindness and way of communication of **NAME of SERVICE** team with us during implementation of service.

1 – Strongly Disagree

2 – Somewhat Disagree

3 – Neither Agree nor Disagree

4 – Somewhat Agree

5 – Strongly Agree

2. How would you evaluate the procedure for implementation and use **NAME of SERVICE**?

   1 – Difficult

   2 – Quite difficult

   3 – Moderate

   4 – Quite easy

   5 – Easy

3. Did you participate on any kind of training organised by **NAME of SERVICE** team for potential users of **NAME of SERVICE** service?

   Yes

   No

4. If you answered yes to the previous question, did you receive all knowledge that you needed for implementation of this service during the training?

   Yes

   No

   If no, what was missing?

5. Please evaluate the quality and availability of information in **NAME of SERVICE** tutorial(s):

   0 – No opinion, I have not used a tutorial

   1 – Very weak (hardly no information is available)

   2 – Weak (only some information is available)

   3 – Average (most of information is available)

   4 – Good (the information is quite easy to identify and learn)

   5 – Very good (the information is very easy to identify and learn)

6. If you answered on previous question options 0, 1 or 2, please tell us what was missing?

7. After implementation of the service, please evaluate the availability of support offered with **NAME of SERVICE**:

   0 – no opinion, I have not used the support

   1 – No support

   2 – Significant delays, sometimes no channel is available, no SLA

   3 – Acceptable availability of channels, acceptable delays, no SLA

4 – Some (but not annoying) delays in support, no SLA

5 – Multiple channels available, SLA available

8. After implementation of the service, please evaluate availability of the SLA support offered with **NAME of SERVICE**:

   0 – No opinion, I have not used the support with SLA

   1 – Very weak/no SLA services (usually failing to meeting agreed conditions)

   2 – Weak (one level SLA, sometimes not meeting agreed conditions)

   3 – Average (one level of SLA, usually, meeting agreed conditions)

   4 – Good (some levels of SLA are available, sometimes not meeting agreed conditions)

   5 – Very good (various levels of SLA are available, always meeting agreed conditions)

9. Are you satisfied with the current support offered for **NAME of SERVICE,** or you would suggest a new approach?

10. Do you think you needed expert knowledge for using this **NAME of SERVICE**?

    Yes
    No

11. How many people participated in your NREN in the process of implementation of this service? (if there are more than one person, please briefly describe their role or hierarchy if there was any.)

12. Approximately how much time did you need for successful implementation of **NAME of SERVICE**?

13. Do you have any suggestions or recommendations based on your experience how to improve **NAME of SERVICE** support, we really appreciate your feedback to help us improve the service even further!

14. If we can contact you for some additional information, please be so kind and leave us your full name and e-mail address.

Appendix D System Usability Scale survey

The System Usability Scale (SUS) provides a “quick and dirty” but reliable tool for measuring usability. It consists of a ten-item questionnaire with five response options for respondents; from Strongly Agree to Strongly Disagree.

*On scale from 1 (Strongly Disagree) to 5 (Strongly Agree), please rate how familiar you with each statement about **NAME of SERVICE**.*

- I think that I would like to use this system frequently.
  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree
- I found the system unnecessarily complex.
  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree
- I thought the system was easy to use.
  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree
- I think that I would need the support of a technical person to be able to use this system.
  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree
- I found the various functions in this system were well integrated.
  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree
- I thought there was too much inconsistency in this system.

1 – Strongly Disagree
2 – Somewhat Disagree
3 – Neither Agree nor Disagree
4 – Somewhat Agree
5 – Strongly Agree

- I would imagine that most people would learn to use this system very quickly.

  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree

- I found the system very cumbersome to use.

  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree

- I felt very confident using the system.

  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree

- I needed to learn a lot of things before I could get going with this system.

  1 – Strongly Disagree
  2 – Somewhat Disagree
  3 – Neither Agree nor Disagree
  4 – Somewhat Agree
  5 – Strongly Agree

References

[Brooke1996]	Brooke, J. "SUS: a "quick and dirty" usability scale". In P. W. Jordan, B. Thomas, B. A. Weerdmeester, & A. L. McClelland. Usability Evaluation in Industry. London: Taylor and Francis, 1996
[Dowd2006]	Mark Dowd, John McDonald, Justin Schuh, "The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities", Pearson Education, 2006 – https://books.google.pl/books?id=t2yA8vtfxDsC
[Findbugs]	http://findbugs.sourceforge.net/
[GITrepository]	https://code.geant.net/stash/projects/SA4T1
[ITIL]	http://wiki.en.it-processmaps.com/index.php/Main_Page
[ITILtransition]	ITIL service transition Second ed., 2011 ISBN: 9780113313068
[Kan2002]	Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (Second ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[Lewis2008]	William E. Lewis, "Software Testing and Continuous Quality Improvement", Third Edition, CRC Press, 22.12.2008
[Meyer2009]	J. F. Meyer, "Defining and Evaluating Resilience: A Performability Perspective", Proceedings of the International Workshop on Performability Modeling of Computer and Communication Systems(PMCCS), 2009
[OWASPtop10]	https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[OWASP]	https://www.owasp.org
[PLM]	http://www.geant.org/News_and_Events/CONNECT/Documents/ CONNECT_16.pdf#search=PLM
[QABestPractices]	GN3 Quality Assurance Best Practice Guide 4.0 http://geant3.archive.geant.net/Media_Centre/Media_Library/Media%20Library/GN3-09-184_Quality_Assurance_Best_Practice_Guide_4%200.pdf
[RIPS]	http://rips-scanner.sourceforge.net/
[SDBestPracticesGuide]	GN3 Software Documentation Best Practice Guide http://geant3.archive.geant.net/Media_Centre/Media_Library/ Media%20Library/GN3-12-330_Software_Documentation_Best_Practice_Guide_2.0.pdf
[SonarQube]	http://www.sonarqube.org/
[STIGchecklists]	http://iase.disa.mil/stigs/Pages/index.aspx
[W3CAT]	Accessibility testing http://www.w3.org/wiki/Accessibility_testing

Glossary

Acceptance Testing	Testing performed to determine whether or not the system has met the requirement specifications.
Accessibility Testing	Verifying that the users with impairments or disabilities can use the system with relative ease.
BC	Business Case is a decision support and planning tool that projects the likely consequences of a business action (ITIL).
Big Bang Testing	Integration testing where all components are integrated are tested at once; eliminates partial integrations and tests, but delays discovery of bugs and makes their isolation and interpretation more difficult.
Black Box Testing	Testing based on the specification, requirements or expectations, without using any knowledge about the system or component internal implementation.
Bottom Up Testing	Integration testing where the lower level components are tested first and then used to test the components that rely upon them.
Boundary Testing	Testing focused on the boundary or limit values and conditions of the software being tested.
Bug	A defect in a software which causes it to perform in an unintended or unanticipated manner due to developer’s inadequate implementation of behaviour.
CBA	Cost/Benefit Analysis, conforming and replacing the Business Case in GÉANT Product Lifecycle Management.
Clear Box Testing	Security testing with full access to the analysed system and full knowledge about it. Also: Crystal Box, White Box Testing.
CSI	The ITIL Continual Service Improvement (CSI) process uses methods from quality management in order to learn from past successes and failures
CVSS	Common Vulnerabilities Scoring System, the value between 0 and 10 describing severity of a publicly disclosed security vulnerability according to a number of criteria. CVSS more than 6.9 denote "high" severity. CVSS less than 4.0 is "low". The rest CVSS scores denote "medium" severity.
Code Inspection	Formal code review where the developer reviews the source code with a group who ask questions analysing the program logic, as well as code against the checklist of common programming errors and coding standards.
Code Walkthrough	Peer review of the source code where the developer leads the review and others ask questions and spot possible issues and errors against standards and other issues; may be performed statically the code and data may be traced with a small set of test cases.
Compatibility Testing	Non-functional testing whether the system is can be matched with the some parts of its environment or other systems.
Conformance Testing	Testing whether the system meets some specification or a formal standard.
Defect	An imperfection, lack or non-conformance to requirements, specification or expectations that causes inadequacy or failure.
Dependency Testing	Examining an application's requirements for an existing software, initial states and configuration in order to maintain its functionality.
Error Handling Testing	Testing concerned with faults or run-time errors that are usually caused by adverse system parameters or invalid input data.

Failure	Lack of success in delivery of intended function, service, purpose or expectation, usually caused by a defect, other error or related failure.
False positive	A situation where an error or vulnerability is mistakenly reported (e.g. by a code analyser) but actually everything is all right.
Fault	Manifestation of error during software execution; may be caused by its design, bugs, system parameters, input or external failures.
Gray Box Testing	Testing a system against its specification but using some knowledge of its internal workings in order to validate some elements of the design, possibly by accessing or manipulating the internal interfaces or data repositories.
HTMLI	HTML Injection, a security vulnerability where an attacker is able to inject an HTML code to the application that will be executed in the victim's browser (a client-side attack).
ITIL	Information Technology Infrastructure Library. A set of practices that focuses on aligning IT services with the needs of business.
Integration Testing	Testing whether an assemblage of several components works correctly.
Load Testing	Non-functional testing by investigating system behaviour on large but attainable workload or number of users in order to estimate response times, throughput rates, resource utilisation, bottlenecks and maximum load and improve overall system configuration and tuning.
KPI	ITIL Key Performance Indicators are used to assess if the processes of an IT organisation are running according to expectations.
Manual Testing	Testing that is carried out manually in order to find defects, without the usage of test tools or automation scripting.
OS	Operating System, OS Command Injection – ability to inject a command to be executed by the operating system being attacked.
OWASP	Open Web Application Security Project
OWASP TOP 10	A list of top 10 categories of security errors in web applications.
Penetration Testing	Black box or white box testing in which an authorised attempt is made to violate specific system security or integrity constraints or policies.
Performance Testing	Non-functional testing conducted to evaluate the compliance of a system or component with specified performance requirements under a particular workload and to identify performance bottlenecks.
PLM	The Product Lifecycle Management defines a complete service lifecycle within the GÉANT environment.
Recovery Testing	Non-functional testing performed to determine how quickly the system can recover after the system crash or failure, by intentionally causing the failure.
Regression Testing	Retesting of a previously successfully tested features or functions of the system in order to ensure that new defects were not introduced or uncovered by its later modifications.
Requirement	Necessary capability, feature, functionality or constraint that the system must meet or be able to perform.
Resilience Testing	Non-functional testing of how well the system is able to recover from disastrous environment events, crashes, hardware and infrastructure failures and similar problems.
Sanity Testing	Unscripted Smoke Testing.

SAC	Service Acceptance Criteria, a set of criteria used for service acceptance testing to ensure that an IT service meets its functionality and quality requirements and that the service provider is ready to operate the new service when it has been deployed.
SCT	Secure Coding Training – an annual three-day workshop where GÉANT developers are taught how to create source code that contains as few security vulnerabilities as possible
Scalability Testing	Performance testing focused on how the system under test handles increases in workload by scaling up (increasing concurrency or size) or scaling out (distributing to newly allocated resources).
Scenario Testing	Approach to testing of complex systems that is not based on Test Cases, but on working through realistic and complex stories reflecting user activities.
SDP	ITIL Service Design Package, the central reference point for all documentation of a service.
SDT	Software Development Team, people who have written the application that is assessed by SA4 T1.
Security Testing	Non-functional testing whether the system meets the specified security related requirements or expectations in order to protect its data and maintain functionality.
Smoke Testing	A quick and simple test of major functional elements of the system to determine whether it basically works.
SOM	Service Operation Manager, a manager of IT operations for a given service.
SQLI	SQL Injection, a security vulnerability where an attacker is able to inject a custom database query or command to be unintentionally executed by the application being attacked
SSL/TLS	Secure Sockets Layer / Transport Layer Security – a set of cryptographic protocols encapsulating HTTP transmission in order to assure confidentiality and integrity of the exchanged data.
Stress Testing	Performance testing to verify system or component reliability under excess workload in order to determine when it will fail and how, conducted by submitting the system to the increasing load or by limiting supply of some operational resources.
SVT	The objective of ITIL Service Validation and Testing is to ensure that deployed Releases and the resulting services meet customer expectations, and to verify that IT operations is able to support the new service.
System	Device, software application, physical infrastructure, product, service, their purposely built composition, construct or design that is the subject of testing; also known as System Under Test (SUT).
System Testing	Black box testing of the whole system from the end user perspective in order to validate whether it meets the requirements.
Test	Test Case, Test Procedure, Test Suite, several Test Suites, or their actual execution. 'Test' should be used only if the intended meaning is obvious from the immediate context.
Test Bed	See Test Environment.
Test Case	Description of inputs and/or conditions, procedural instructions and expected results of Test Execution, in the form of brief narrative summary or as a detailed specification. One of several Test Cases establish the

	criteria that need to be met in order to consider some specific Requirement as satisfied or feature as working.
Test Case Specification	A detailed elaboration of a Test Case, provided in a Use Case or Test Suite specification, a document that details all Test Cases, or in a separate document dedicated to a single Test Case.
Test Environment	Test execution environment configured for testing. Also Test Bed.
Test Execution	Process of executing the Test Cases, Test Suite or scenario (in Scenario Testing) and comparing the expected and actual results.
Test Item	The individual element, component or subsystem of the tested System.
Test Log	Record of Test Cases execution and obtained results, in the order of their running.
Test Procedure	Detailed instructions and sequence of steps to be followed while executing a group of Test Cases (such as a Test Suite) or single Test Case. Sometimes referred to as Test Scenario.
Test Scenario	Test Procedure, Test Case scenario, or scenario used in Scenario Testing. This term should be used only if the intended meaning is obvious from the immediate context.
Test Script	Sequence for instructions that need to be carried out by a tester or an automation test tool on the System in order to execute a Test Case, or test a part of System functionality.
Test Suite	Collection of Test Cases that are related to the same testing work in terms of goals and associated testing process.
Test Tool	Software used in the testing of a System, its component, or documentation.
Testing	Process of exercising a system or its components to verify that it satisfies specified requirements and to detect errors.
Top Down Testing	Integration testing where the higher level components are tested first with simulated lower level components and then used to facilitate the testing of the actual lower level components.
Traceability Matrix	Table or standalone document for tracing Requirements at different levels (Business Requirements, User Requirements, Functional Specification, Design Specification, implementation/code, Test Plan, Test Design Specification, Test Cases, defects/issues) or for linking Requirements and Test Cases.
UML	Unified Modelling Language
Unit Test	Testing of an individual module in order to determine that it works as intended by its developer.
Usability Testing	Non-functional testing performed to measure how easily end users can learn and use the System.
Use Case	High level description of a specific system usage, or set of behaviours or functionalities, specified from the end-user perspective. This is not to be mistaken for UML use case or related to UML use case testing.
User Acceptance Testing	Formal system evaluation performed by a user or customer in order to determine whether the system meets all previously agreed requirements.
Volume Testing	Non-functional testing where the software is subjected to a huge volume of data or which confirms that any items that may become large over time (such as counts, logs, and files) can be accommodated by the software and will not cause it to stop working or degrade its operation.

Walkthrough	A review of requirements, design, system, software code or documentation guided by its author.
White Box Testing	Testing based on an analysis of internal design and implementation of the system.
XSS	Cross-Site Scripting, security vulnerability where an attacker is able to inject an active script code (e.g. JavaScript) to be executed in the victim's browser, usually intended to steal the victim's session identifier in order to impersonate the victim.