# Definition

## of the Testing and Evaluation Framework for the Virtual Company Dossier System

## Log of Changes

| Version | Date of Change | Changed By | Summary of Change |
|---|---|---|---|
| 0.1 | 17.01.2011 | Daniel Reiser | Initial version |
| 0.2 | 24.01.2011 | Daniel Reiser | Additional sections on test organization, test groups and responsibilities |
| 0.3 | 31.01.2011 | Daniel Reiser | Changes to Test organization |

## Purpose

This document defines the framework for the testing of the Virtual Company Dossier System Pilot. Covered by this definition are the following topics:

- Definition of the testing approach, different test types, test execution and documentation
- Identification and description of different test scenarios and test cases
- Definition of responsibilities
- Definition of time schedule

## Related resources

| # | Resource title | Reference | Author(s) |
|---|---|---|---|
| | Test Framework Overview | PowerPoint presentation | Ansgar Mondorf |
| | BSCW root folder for testing and evaluation | BSCW folder | |
| | Application Profiling and Generic Test System Introduction | Slideshow (PDF) | Daniel Reiser |
| | Context/Value Association using Genericode 1.0 | OASIS Public Review Draft | OASIS |

| | | | |
|---|---|---|---|
| | VCD CodeLists | VCD CodeList Spreadsheet | WP2 |
| | Evaluation guideline for European VCD System | Word document | WP2 |

# Content



# List of tables

## List of figures

## Introduction

The testing framework defined in this paper is providing a structured approach of testing the software distributions developed for the VCD system. It contributes to the internal need for testing the WP2 software as well as the overall PEPPOL target of reaching maturity milestones. It should be used to verify that implementations comply with initial specifications and requirements. Conforming implementations are a necessary prerequisite for achieving interoperability among implementations. To support the definition, execution and reporting of tests, several tools can be used. Details on this will be stated in the relevant paragraphs.

## Components of the Testing Framework

Figure 1 depicts the overall testing framework. Two main test groups have been identified. For each of them, specific test types as well as tools to support and document test execution have been defined.

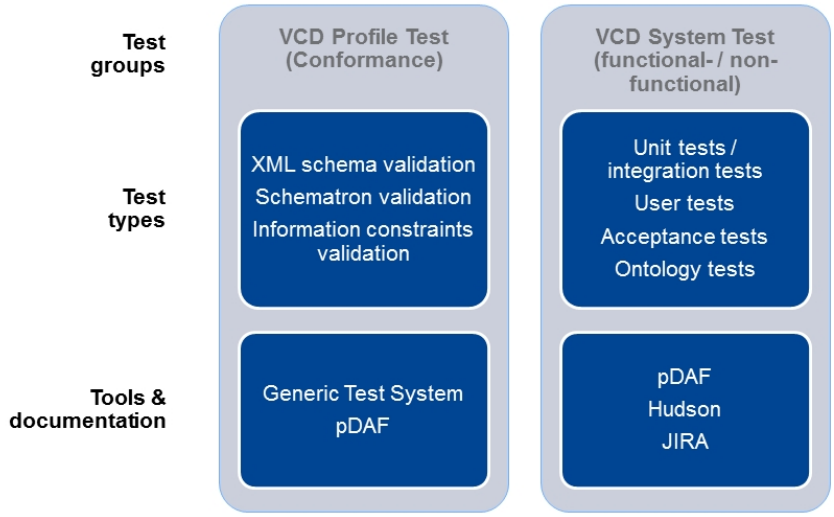



*Figure 1: Overview of the Testing Framework*

### Test Group 1: Conformance testing

The aim of VCD conformance testing is to test whether software components are able to generate and understand correct document instances. Document instances are correct if they are valid regarding:

- Well-formed XML structure

- Mandatory / optional elements and cardinalities
- Element data types
- Element values (code lists, value patterns: e.g. URI, UUID, etc.)
- Semantic validity
- Further information constraints (e.g. file reference checks)

These validation objectives shall ensure that both technical and semantic conformance of VCD instances can be tested and ensured. Hence, both the technical structure and additional semantic information constraints and business rules have to be defined in order to validate document instances against these rules. These rules shall cover the aspects described in Table 2.

| Aspect | Description |
|---|---|
| **Document structure** | Correctness of the structure and data elements of an XML instance as defined in the corresponding XML schema |
| **Element cardinalities** | Optional elements MAY exist, mandatory elements MUST exist. Prohibited elements will not be part of a profile XML schema.<br>- 0..1 = optional, zero or one occurrence<br>- 0..*= optional, zero or more occurrences<br>- 1 = mandatory, one occurrence<br>- 1..* = mandatory, one or more occurrences |
| **Data types** | String, integer, float, date, time, boolean |
| **Value ranges** | - Length (for elements of type string)<br>- Range (for elements of types integer, float, date and time) |
| **CodeLists** | Correct use of values as defined in CodeLists |
| **References** | - Correct file references in a document (cbc:FileName)<br>- Existence of VCD sub folders for economic operators defined in the VCD Package meta-data file (cac:VCDReferenceID).<br>- Correct element references (cbc:ProvesCriterionID, cbc:ProvingEvidenceID) |
| **Value patterns** | Correct values and format of specific element values, e.g.:<br>- UUID: UUID version 4 (random UUID)<br>- Date: yyyy-mm-dd<br>- Time: hh:mm:ss |

*Table 1: Validation aspects of VCD conformance testing*

### Test types

The test types of the VCD conformance testing group cover the validation objectives listed above. Table 1 describes the three test types of this group.

| Test type | Description | Rationale |
|---|---|---|

**Kommentar [MK1]:** This section misses validation of Digital Signatures, ZIP-Package-Format (not only the XML-part), … And which tools are needed to test these parts?

**Kommentar [pm2]:** In relation to comment MK1. I must add that the VERIFICATION of Digital signatures applied to any document forming the "DocumentReference:Attachment" is an independent process that implies the use of a specific technology. This process is part of the VCD BUILD activity. As a consequence of that, the final VCD instance, shows a dedicated data structure, i.e. the "ResultofVerification". The Signature VERIFICATION and the VCD instances VALIDATION are considered so far different processes for different purposes.
Anyway,

| | | |
|---|---|---|
| **XML schema validation** | XML schema validation of VCD instances against the specified XML schemas. | VCD instances have to be conformant to the VCD XML schema as a minimum requirement of validation. |
| **Schematron validation** | Schematron validation of VCD instances against Schematron rules. | Used for additional validation against rules not covered by the basic schema validation. |
| **Information constraints validation** | Validation of VCD instances against additional information constraints derived from business rules. | It is necessary to test such constraints that cannot be covered by XML schema and Schematron validation, e.g. file references within an XML instance. |

*Table 2: Conformance test types*

### Test objects

The test objects being validated in the Conformance test group are:

- VCD-PreSkeleton (T-Skeleton, TC-Skeleton) and VCD-Skeleton (TCE-Skeleton): Meta-data files and folders.
- VCD and VCD-Package: Meta-data files and folders.
- VCD-Container: Zip file, meta-data files and folders.

Document instances created and processed by software components must conform to the technical structure and rules defined by the corresponding XML schema definition, Schematron rules as well as additional rule definitions of the Generic Test System.

### Testing approach

Figure 2 illustrates the testing approach and artefacts used for conformance testing. These steps shall allow a structured and organized testing of VCD document instances. Details are provided in the following sub-sections.

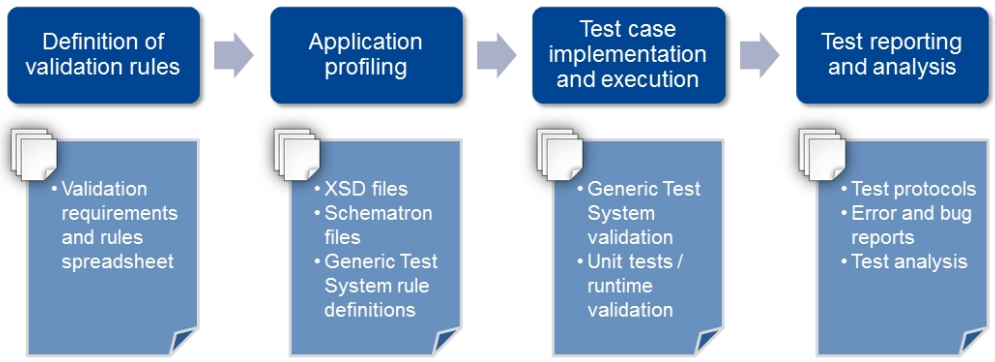



*Figure 2: Conformance testing approach and artefacts*

### *Definition of validation rules*

This first phase covers the definition of validation requirements, business rules and information constraints that define the rules and constraints against which document instances are being validated. The validation requirements spread sheet is intended to document all these requirements and rules.

### *Application profiling*

Based on the validation rules defined in the previous phase, application profiles (also known as domain profiles) are created in order to provide the technical basis for implementing the tests. The application profiles will be defined according to the phase of the VCD generation process, covering

- T-profile,
- TC-profile,
- TCE-profile,
- VCD-profile,
- VCDPackage-profile and
- VCDContainer-profile.

These profiles shall provide a logical grouping according to the VCD generation phase. For each of them, XML schemas, Schematron rules as well as additional validation rules shall be defined. This is necessary as different domains or applications, in which VCD documents are processed, have (slightly) different rules regarding allowed data elements, cardinalities and information constraints (e.g. evidence documents are not foreseen to be part of a T-Skeleton).

For the detailed definition of validation requirements and rules, the primary focus should be on the TCE-Skeleton, VCD and VCD Package level. However, to ensure that valid T- and TC-Skeleton instances are exchanged during the NVS-EVS interaction, a sub-set of schema and Schematron validation shall be derived for the T- and TC-profile.

The step of application profiling shall lead to a concrete set of validation rules and test system configurations that can be processed by validation software. Therefore, it is necessary to document all rules in a common way using existing standards and methodologies. Table 4 describes the validation artefacts to define and document the validation rules.

▪•●

| Artefact | Description |
|---|---|
| **XML schema files** | Specifies a sub-set of the full VCD data model specification. It defines the technical structure of a document instance, including allowed elements, cardinalities, data types and value ranges. |
| **Schematron files** | Defining Schematron validation rules. |
| **Context/value associations (CVA) files** | Expressing the relationship between controlled vocabularies (e.g. CodeLists) and document elements. |

| | |
|---|---|
| **Generic Test System rule definitions** | Defining validation rules for the Generic Test System. |

*Table 3: Validation artefacts of VCD conformance testing*

For each of the aforementioned application profiles, these validation artefacts shall be created (depending on a particular profile it may be sufficient to only create some of these validation artefacts).

### *Test case implementation and execution*

For the actual testing of VCD documents, each application profile and corresponding validation have to be implemented. There are three different types of possible implementations according to the test scope:

1. Unit tests: Validation and tests performed during development to ensure that a unit of software produces correct document instances.
2. Runtime validation: Validation performed during runtime to validate the document instances that are created (output) or read (input) by software components.
3. Generic Test System validation: Manual validation using a specific configuration of the Generic Test System.

**Kommentar [MK3]:** Has this to be done manually?

Unit tests and runtime validation in the narrow sense shall be part of the functional testing of software components (see section "Test Group 2: Functional and non-functional system testing"). Hence, runtime validation should be a feature that is built into software components to avoid errors caused by inputting or transferring wrong data. Unit tests shall be defined and executed to ensure that a software component that is processing VCD instances has been implemented correctly.

Generic Test System validation refers to validation triggered by testers using a configured Test System. The Generic Test System allows configuring test systems (so called Test System Configurations), which provide an executable file that opens a graphical user interface through which the validation process can be started by providing a document instance of the test object. Document instances, such as VCD Packages or VCD Container, can be selected via this GUI in order to execute the tests defined in the test system. Detailed reports will be generated during the execution and provided as HTML output by the tool.

### *Test reporting and analysis*

The results of test runs have to be documented in a common way. Hence, test protocols need to be created for the different test runs. In case of automatic unit tests, the reporting capabilities of the test execution system can be used (e.g. the Hudson test summary). For manual tests using the Generic Test System, detailed test protocols as well as test results shall be stored in a central system. The pDAF testing tool is an appropriate tool to keep track of all the performed tests as well as to report and document their execution and results. In addition to the pDAF testing tool, a bug tracking system shall be used to report the errors and possible bugs explored during the test runs.

Based on the results of test execution and error reports, a test analysis should lead to decisions on how errors or bugs can be fixed as well as which functionality has to be altered or added to solve a reported bug.

#### Testing suite and artefacts

The testing suite for conformance testing contains some software tools to create, document and store the different validation rules, Schematron files and test system specifications. For the collection of validation requirements and rules, excel spread sheets will be used. These will be stored in BSCW in the corresponding folder. Schematron files are created to define the Schematron validation rules, the configurations of the generic test system will also be stored there. The pDAF platform shall be used as a general test management platform to document, maintain and report test cases and test executions.

## Test Group 2: Functional and non-functional system testing

This test group covers all functional and non-functional software testing. Functional tests are intended to ensure that the implemented functionality complies with the specifications and that all requirements are met. Non-functional testing shall cover all user and acceptance tests to ensure that the software is working well and has a good level of usability.

Functional tests cover testing of the developed functionality. The testing approach should be in line with the Scrum approach which has already been followed by the development team. This affects primarily the definition of test cases and bugs in JIRA for each of the defined user stories. For detailed definitions and documentation of test cases, the pDAF testing platform shall be used. The definition of user stories and test cases as well as a structured approach to execute and document the test cases shall ensure quality of the delivered software components.

To enlarge the functional testing and to ensure a good usability and user acceptance of software components, additional non-functional and acceptance tests should be defined and executed. This means, that the developed software has to be tested from the end user's point of view, including testing of graphical user interfaces, usability and performance.

To also address legal requirements that have to be covered by the VCD, the data quality of national and European rule sets that is stored in the ontology has to be tested as well. As the mapping of criteria to evidences is a core feature of the VCD system, ensuring the quality of data this mapping is based on is very crucial.

### Test areas and types

In order to achieve the objectives described before, several different test types have to be performed. Table 5 describes the overall test areas, by which the test types shall be grouped.[1]

| Test area | Description | Rationale |
|---|---|---|
| **Functional: Unit tests / system tests / integration tests / acceptance tests** | - Testing software components against specifications.<br>- Testing of functional units of the software components. After bug-fixes or updates have been implemented, tests have to be re-executed.<br>- Testing software modules once they are integrated with related modules via their interfaces. | It has to be tested that a software component implements all the features that have been defined in the specification. Once a new functional unit is implemented, is has to be tested whether the requirements are met and whether the functionality is performing correct, even after changing and updating a module. After integrating formerly distinct software modules, their well-functioning has been assured after the units are integrated. |
| **Non-functional: User, acceptance, ontology tests** | Testing software ready for a release whether it is working correct. | Non-functional testing in terms of correct working of implemented software applications is crucial before releasing it. As mapping between criteria and evidences is a core feature, ontology testing is another important of this test type. |

*Table 4: Functional and non-functional test areas*

**Kommentar [MK4]:** These are not „non-functional" tests. Usually non-functional tests are e.g. performance tests, system stability tests, fail-over tests, security tests, … (like in the following table).
The Ontology provides "logic" and "functionality" and should therefore not be tested only in the non-functional section.

Each of these test areas is further detailed by several particular test types, which have to be performed. These are defined in Table 5.

**Kommentar [MK5]:** How will they be performed? Are there additional tools necessary? E.g. for performance tests, security tests, etc.

| Test types | Description |
|---|---|
| **Functionality** | Implemented functionality of a software component or unit has to be tested according to correct functioning and requirements. |
| **Process** | Sequence of actions that have to be executed to reach a specific state or target. |
| **Performance** | The test object has to be tested whether its performance (in terms of well-functioning, usability, availability and reliability) is not affected by increased stress or load. |
| **Code review** | Examination of source code focussing on formatting, structure, etc. in order to ensure readability by others. |
| **Documentation** | Completeness and comprehensibility of documentation. |
| **Security** | Checking security aspects, such as data security and data protection. |

[1] The grouping of test areas and test types is based on the test case definitions done in the pDAF test management tool.

| | |
|---|---|
| **Installation and configuration** | Check correctness of installation and configuration of system components. |
| **Regression** | Re-running tests after bug-fixing or new releases to ensure that new or changed functionality does not negatively affect other components or units. |
| **Maintenance** | Post-release maintenance of system components and software |
| **Fall-back and recovery** | Check fall-back and system recovery strategy of system components |

*Table 5: Functional and non-functional test types*

### Test objects

The objects, on which the aforementioned test types will be executed, are primarily software applications, software components and their functional modules:

- VCD Builder
- EVS
- VCD Designer (incl. EVS/NVS interface)
- VCD Viewer
- Ontology

**Kommentar [MK6]:** Can there be a generic way to also test the NVS implementations? This could be very helpful (only for the use cases that are identical in every implementation; only result-testing because the GUI will differ). E.g. test of VCD-Package-ZIP not only regarding "conformance" but also "logic/legal correctness".

### Testing approach

The approach for this test group shall be requirements- and specification driven. This means, that test cases shall be defined according to the functional and non-functional requirements which have been specified for the development of software components. The steps illustrated in Figure 3 shall be completed during the software testing.

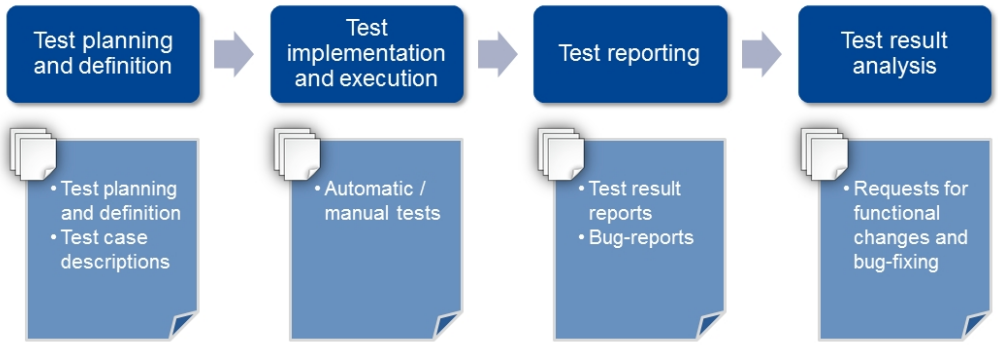



*Figure 3: Software testing approach and artefacts*

#### *Test planning and definition*

Test planning and definition contains the required steps to specify the tests that should be executed. Most important will be the definition of test cases, which describe in detail what is being tested

(software module, user story, requirement, etc.), what are the pre- and post-conditions, expected results, test steps as well as test data being used for test execution.

#### *Test implementation and execution*

In case of manual tests, the test implementation and execution comprises setting up the environment, i.e. configuring the software component being tested, setting up pre-conditions, providing test data, etc., as well as executing the test following the "test protocol" defined in the test case. For automatic tests, a test system such as Hudson will automatically execute the unit tests that have been implemented by developers (e.g. JUnit tests). For automated testing of web applications, the Selenium test framework may be used to test the functionality of graphical user interfaces.

#### *Test reporting*

Test reporting comprises the documentation of test outcomes once a test has been executed. Most important here is the reporting of bugs explored during the test execution. These error and bug reports should be documented in a common and structured way, e.g. by using a bug tracking system (JIRA, Bugzilla, pDAF).

#### *Test result analysis*

Based on the test and bug reports, it has to be analysed and decided, how bugs can be fixed or which functionality has to be changed or added to resolve an error. These results should be used as input for development teams to perform required implementation tasks.

### Testing suite and artefacts

The different phases will produce different kinds of artefacts to define and document the tests. This includes:

- Test plans: Definition of overall planning of the tests, test cases, etc.: Define test cases, including descriptions, pre-/post-conditions, test steps, etc.
- Test data: Definition of test data used to execute the tests.
- Test reports: Documents reporting the results of a test execution.

The testing suite consists of several software tools used to manage and execute all tests. These can be grouped according to the different test phases:

- Test definition, documentation and storing of artefacts:
  - pDAF: General test management and maintenance, definition of test cases.
  - JIRA: Defining user stories for software features being developed.
  - BSCW: Additional document management, if required.
- Bug-tracking
  - JIRA, m2n Bugzilla (PEPPOL.AT): Collect and document discovered bugs.
- Test execution
  - Hudson: Execution of automatic tests (unit tests).
  - Selenium: Execution of automatic tests (GUI testing).
- Test reporting

- pDAF: Documentation and reporting of test executions.
- BSCW: Additional document management, if required.

## Test organization

The test areas to cover the functional and non-functional as well as the conformance testing groups are grouped by each component: [2]

**Kommentar [MK7]:** The EVS should also be part of this list

- VCD Designer
  - Functional / non-functional testing
  - Conformance testing
- VCD Viewer
  - Functional / non-functional testing
  - Conformance testing
- VCD Builder
  - Functional / non-functional testing
  - Conformance testing
- Ontology
  - Non-functional testing

### Functional and non-functional testing

The functional and non-functional testing shall be driven by user stories and D2.2 specification. User story testing shall ensure, that all functionality which has been planned and implemented during the shared components sprints exists and is well functioning. D2.2 specification testing on the one hand shall be specification driven to assure that requirements defined in deliverable D2.2 are met. On the other hand is intended to analyse potential gaps between the existing functionality and the software specification of deliverable D2.2. Results of this gap analysis shall lead to a detailed report on feature requests which have been planned in D2.2 but have not been implemented.

For user story testing, JIRA should be used intensely for several purposes: It defines all the user stories which have been implemented as functionality during the SCD sprints. Some test cases have already been defined for some of the user stories. To follow this common approach, test cases should be defined directly in JIRA as sub-issues of each (testable) user story, briefly indicating what has to be tested for a certain user story. This should be done at least for all those tests which can be performed automatically by implemented JUnit tests.[3] For those test cases, which require actions done by test users, pDAF should be used to define test cases and report on test execution. As pDAF

[2] According to this groups, test categories and test areas have been defined in pDAF. To provide initial templates and samples, some test cases for "VCD Designer: Functional / non-functional testing" have been defined and executed.

[3] These will be executed by Hudson or by developers executing the corresponding maven command during development of a software unit.

also provides better means of managing test cases and defining test case details, this tool should be used to define detailed test cases, including test data, pre-conditions and test execution steps.

If bugs any bugs have been explored, they should be reported as a new “bug” for each user story. If pDAF is being used for a test case, the test execution report should contain references to the created bug reports.

### Conformance testing

Conformance testing is driven by the definition of validation requirements and rules in order to test and ensure that software components are conformant to the specification of the VCD data model. Core of this testing area is the validation of VCD and VCD Package instances created or processed by software components.

Unit tests and runtime validation is primarily a software feature, that has to be implemented in software unit, using the validation rules defined in the first conformance testing step; hence this feature it should be covered by a user story being tested in the functional software testing stream.

The manual conformance testing shall lead to a configured test system that allows manual testing of document instances according to the rule-sets defined for each of the VCD application profiles. According to the different VCD application profiles and the different software components that create or process a VCD instance, a pDAF test case should be created. This test case should at least provide the following information:

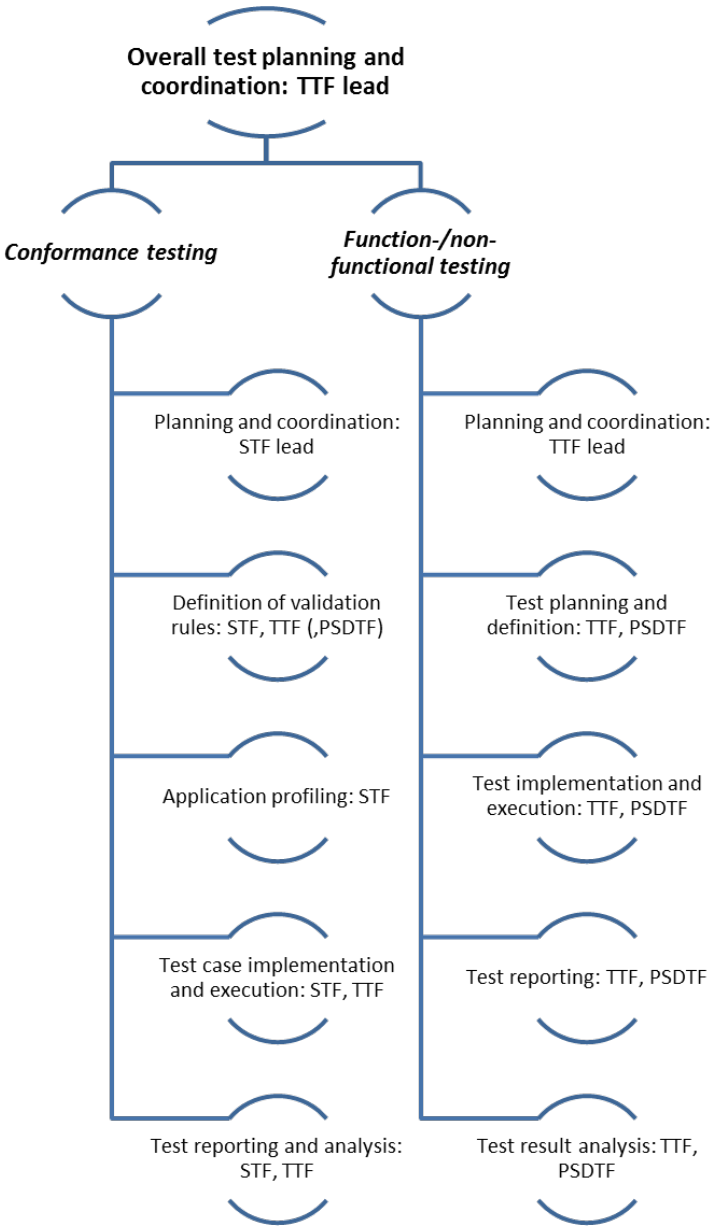
- Summary: A brief summary of the test, indicating which profile is being tested, which software unit produced the document instance, in which process step of generating a VCD has the instance been created, etc.
- Underlying specification: The application profile defining the validation rules.
- Data / System preconditions: A short descriptions of the preconditions that are required to execute the test.
- Test steps: A description of the steps to be performed during test execution, as well as expected results and attachments (such as input data).

After the test case has been executed, the results produced by the generic test system should be attached to this test case in the test execution section (this can be done by storing the html output of the test system and attach it as a zip file). Additionally, the test execution report in pDAF shall also contain a summary about the different kinds of tests as well as the general summary of results, as it is shown by the validation report summary page of the test system’s output.

## Responsibilities

The overall coordination of tests should be responsibility of the technical task force, in particular the technical task forces lead. For the different test types as well as different test phases, several persons

with different backgrounds should be responsible for executing the tasks. The overall responsibility picture is described in the following: [4]




[4] TTF = Technical task force; STF = Schema task force; PSDTF = Piloting, sustainability and dissemination task force

## Next steps

After the V1.0 release of the VCD system components, it is necessary to ensure that all delivered software components implement the desired functionality correctly, provide a sufficient level of usability and are able to read and create valid instances of VCD documents. A detailed testing cycle should be performed as a next step following the Scrum-based sprint approach known from the development of VCD shared components.

According to the main tracks, the following division of tasks can be identified:

- Setting up testing framework:
  - Finalize testing framework proposal
  - Set up test environment, consisting of tools, templates, etc. (Hudson, pDAF, BSCW, etc.)
  - Detailed time planning on testing and evaluation sprint.
- Conformance testing:
  - Defining validation rules for VCD instance validation
  - Defining and describing test cases to test the validation rules, including general test case descriptions as well as definition of application profiles as described above.
  - Implement automatic tests (built-in software features).
  - Set up the Generic test system for performing regular manual tests.
  - Report test outcomes, test analysis.
- Functional and non-functional software testing:
  - Derive and define test cases from specifications and user stories.
  - Implement automatic tests, set up manual tests.
  - Execute tests (manual testing; automatic unit tests will be performed by Hudson automatically once a new feature has been implemented).
  - Document and report test execution, results and bugs.
  - Analysis of bugs and bug-fixing.