# Testing Handbook

## Österreichische Post AG

## Version 1.16

## Contents

# 1 The Purpose of this Document

This handbook contains standards and general regulations for the design and performance of testing activities in the course of all software development projects (in-house, outsourced, combined) in the Austrian Post's IT Department.

It defines and describes testing goals, the testing process, the underlying approach to testing (method, strategy), test tasks, the required testing infrastructure, the resources, means, procedures and tools which are used as well as planning requirements for general testing activities.

In doing so, it specifies which test-relevant issues must be considered during software development projects and also serves as a basis for the development of testing concepts. All of the factors laid out in the handbook must be considered as part of any testing concept and described according to the project's specific content and scope.

This handbook is for the use of all employees who are actively involved in testing activities.

In particular, it is aimed at employees in testing roles who are responsible for test planning, controlling, test design, test preparation, test performance and reporting.

## 1.1 Document Status

| **Scope of Application** | **Classification** | **Status** |
|---|---|---|
| Österreichische Post AG | Internal | 🟩 |

# 2 Version History

| Version Number | Date | Amendment/s | Author |
|---|---|---|---|
| 1.0 | May 2015 | Created | Bernhard Rauter, Robert Licen |
| 1.1 | Dec. 2015 | • Chapter 6 in full<br>• Chapter 8.3 Test Completion Criteria<br>• Chapter 9.4.6 Test Completion Criteria (Acceptance Test)<br>• Chapter on Unit Testing as attachment (now Chapter 12)<br>• Other minor amendments | Norbert Fux |
| 1.2 | May 2017 | • Chapter 8.2 Deviation Workflow extended | Helmut Nitsch |
| 1.3 | Jan. 2018 | • Chapter 7.6 Test Performance (SAP sub-chapter)<br>• Chapter 8.1 Targets<br>• Chapter 8.3 Description of Deviations (SAP sub-chapter)<br>• Chapter 8.4 Classification of Deviations<br>• Chapter 9.2 Test Completion Criteria<br>• Chapter 10.4.6 Test Completion Criteria<br>• Chapter 11 Testing Tools (SAP added) | Peter Pichler |
| 1.4 | Apr. 2018 | • Adjusted and extended regarding software quality in Agile environments | Andreas Mayerhofer-Bollek |
| 1.5 | May 2018 | • Incorporation of test automation concept | Andreas Mayerhofer-Bollek |
| 1.6 | Jun. 2018 | • Testing Roles expanded | Andreas Mayerhofer-Bollek |
| 1.7 | Jul. 2018 | • Test Environments chapter extended | Norbert Fux |
| 1.8 | Aug. 2018 | • Staging documentation in Test Environments chapter extended | Norbert Fux |
| 1.9 | Mar. 2019 | • Role Descriptions revised | Norbert Fux |
| 1.10 | Mar. 2019 | • Review, 13.6 SAP Testing Tools removed | Peter Pichler |
| 1.11 | Apr. 2019 | • Testing Tools updated | Peter Pichler |
| 1.12 | Aug. 2019 | • Creation of Chapter 9 Test Data Concept | Frank Kofler |
| 1.13 | Feb. 2021 | • Checklist for (new) projects added<br>• Test Automation restructured and extended<br>• JMeter and Appium added as tools<br>• TM Role Descriptions revised<br>• Troubleshooting + formatting adjustments | Peter Fußl and Bernhard Havel |

| | | | |
|---|---|---|---|
| 1.14 | Jul. 2021 | • Test Data Generators adjusted<br>• KPI Test Coverage extended<br>• Software version numbers removed<br>• Chapter 11.2 Deviations adjusted<br>• Chapter 6.4 Integration Tests extended | Erik Fischelschweiger |
| 1.15 | Aug. 2021 | • KPI Test Coverage adjusted<br>• RACI Matrix revised | Erik Fischelschweiger, Bernhard Havel |
| 1.16 | Aug. 2022 | • Updated Additional Documents | Erik Fischelschweiger, Michael Zach |

# 3 Background Documents

| ID | Document | Version | Link | Author |
|---|---|---|---|---|
| 01 | IEEE Standard for Software Test Documentation (ISO/IEC/IEEE 29119 Software Testing) | | | Institute of Electrical and Electronics Engineers |
| 02 | Software Quality Guidelines | 4.0 | https://postat.sharepoint.com/sites/wg_270/Documents/Software_Qualit%C3%A4tsrichtlinien.pdf?csf=1&cid=6105a95a-e49c-4cab-82e3-b4ef0ad951b3 | Gregor Neff |

## 3.1 Additional Documents

The following documents assist in the testing process and test management. They can be accessed via the TQM SharePoint.

| Document | Test Phase | Purpose/Use |
|---|---|---|
| Testing_Concept_Template | Test Planning | Documentation of the testing procedure in a programme (master test concept) or project (test concept). Based on ISO/IEC/IEEE 29119. |
| Test_Status_Report_Template | Controlling | Regular reporting to stakeholders. |

# 4 Testing Procedures at the Austrian Post

By way of introduction, below is a summary of key points for projects undertaken at the Austrian Post.

## 4.1 Checklist for New Projects

The checklist below serves as a starting point for any project activity: it is not only for use during performance once testing has begun. It must be observed in this form. It primarily serves as a guide for new projects at the Austrian Post and must be adhered to.

### 4.1.1 Before Project Tendering/Start

- ❑ Queries from/involvement of a Test Manager.
- ❑ Cost estimates for testing need to be prepared and coordinated with the responsible Test Manager (manual, automated, and integration tests).
- ❑ It has been clarified which in-house team will take over the project on completion.

### 4.1.2 Project Tendering

- ❑ The *Testing Handbook* and *Software Quality Guidelines* have been handed over (and accepted).
- ❑ The Austrian Post‘s Test Automation Framework must be used for test automation (including the linking of test cases in Azure DevOps).
- ❑ The target KPIs for quality and testing (e.g., rate of test automation, degree of test automation, unit test code coverage, etc.) have been communicated.
- ❑ Azure DevOps (repositories, pipelines, ARM templates, ticket system, etc.) must be used.
- ❑ Azure DevOps artifacts must be created for the project lifecycle in a way that facilitates reusability and maintenance (including transfer to internal Austrian Post teams/Operations) – any regression tests must be added.
- ❑ The staging concept must be applied as per the Austrian Post’s guidelines.
- ❑ SonarQube must be used for statistical code analysis.
- ❑ The Austrian Post’s and the SAFe Framework’s set time periods (“Sprints”) are to be adjusted accordingly.

### 4.1.3 General

- ❑ Quality and testing are part of the 360-degree project review.

#### 4.1.3.1 Azure Set-Up for Tickets (User Stories, Bugs, Test Cases, etc.)

- ❑ In Azure DevOps, all of the tickets (epics, features, user stories, bugs) from all of the teams involved in a project use the same area path (that of the project) – the ‘iteration path’ refers to the team/sprint level.
- ❑ User Stories which cannot be tested (e.g., for analyses, workshops, etc.) are tagged as “not testable” or “not test relevant”.*
- ❑ User stories must be marked as “automatable”.

   - Either: Manual test cases are marked with the tag “automatable”; after performance, they are given the automation status “automated”.
   - Or: They are marked with a tag as tasks under the particular user story.
   - In any case, it must be ensured that the KPIs (degree of automation, rate of automation, etc.) can be ascertained using Azure DevOps Queries.

- ❏ The user story/test case/bug work items that are used must be linked correctly in order to ensure that test coverage and traceability can be evaluated.
- ❏ The results from the performance of automated test cases must be documented in Azure DevOps (creation of test cases in Azure DevOps and linking with tests in Post.TAF).

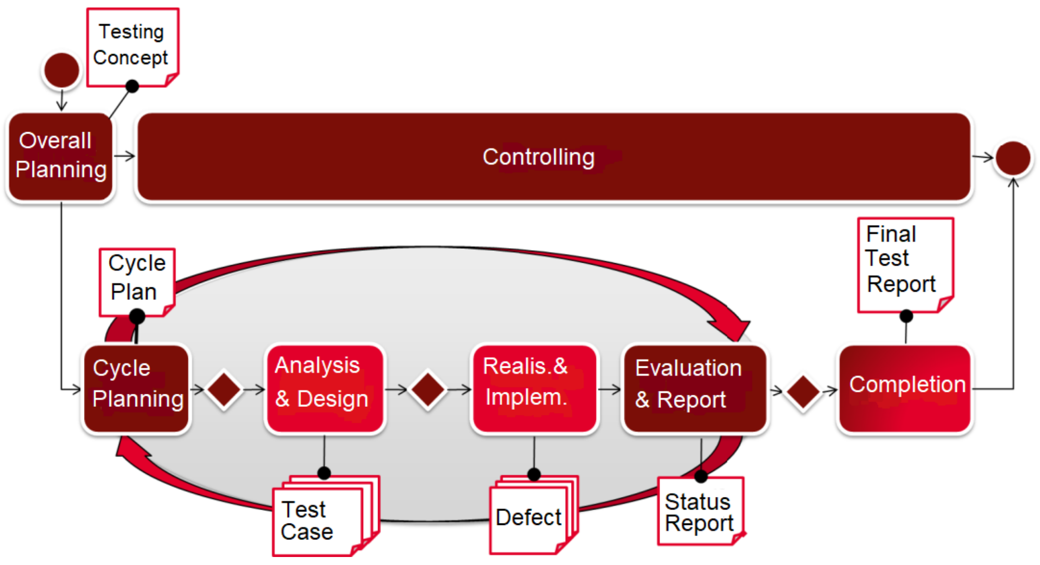
**This should not actually occur where user stories are used correctly (this should be shown using Tasks, for example)*

# 5 The Testing Process

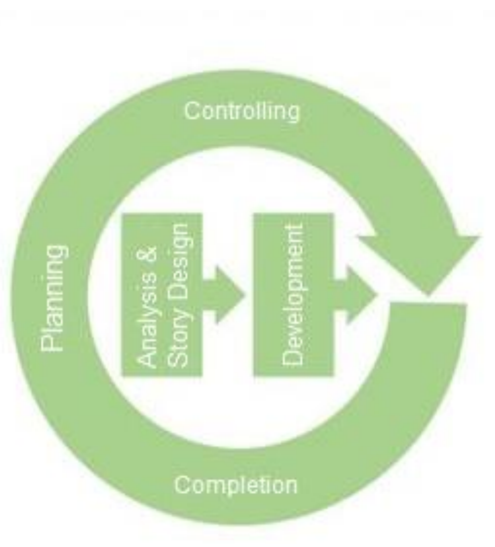
## 5.1 Phases at a Glance

Software testing is based on a repeatable process. To coordinate this testing process with the development process optimally, a distinction is drawn between the following phases as part of the testing process in the different developmental approaches:

**Traditional Testing Process**:



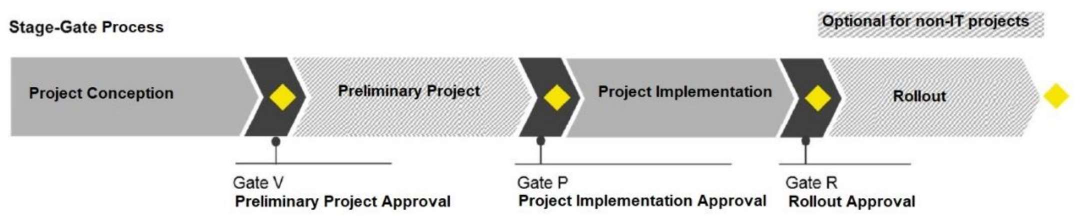
**Agile Testing Process:**



- **Test Planning**
  A distinction is made between (1) the initial overall planning for an undertaking (programme or project) as part of a traditional approach or "Sprint 0" as part of Agile development projects, and (2) regular sprint planning.

- **Analysis & Design**
  Test design techniques are applied for the purpose of analysing (= reviewing) existing basic test documents (specifications, use cases, user stories, etc.) and, based on these, the derivation (= design) of test cases.
- **Implementation & Performance**
  Automated test scripts and testing aids (test data, testing tools, test environment) are implemented and preparatory test cases performed; the performance is logged and the deviations which have occurred documented.
- **Evaluation & Reporting**
  The results from a test cycle or a sprint are made available. As a result, transparent reports on the progress which has made are provided on an ongoing basis.
- **Controlling**
  In a traditional approach, the Test Manager continuously coordinates the test team and is responsible for controlling test projects (effort, costs, risks). The team is responsible for this in an Agile approach. It is practised according to the 'Pull Principle'. If required, controlling can be introduced across the board.
- **Completion**
  Relevant KPIs are made available; test cases and testing aids are reassessed and archived; the final test is communicated.

## 5.2 Embedment in the Stage-Gate Process



1. Testers or the test team are involved as early as possible during the preliminary project. Testers can already help during the development of acceptance criteria for requirements. They carry out initial quality assurance by reviewing requirements. Defects which can already be highlighted and rectified at this early stage incur the lowest possible costs. In the process, general comprehensibility, testability and ambiguous or incomplete descriptions are examined.

2. A rough indication is formed on the basis of the requirements that have been submitted. A test strategy is recommended on the basis of a risk assessment.

3. **In a traditional approach**: The next step sees the Test Manager put together a detailed plan for testing activities. Project members are nominated from within the test team and assigned work packages. If required, further testing specialists are brought in via framework contracts with external service providers (via Supplier Management at the Austrian Post).

**In an Agile approach**: Detailed planning is carried out by the implementation team before each sprint as part of sprint planning.

**In both approaches:** During the initial planning stage, requirements with respect to infrastructure (environments, test data, tools, testing equipment) are determined and purchasing/organisation initiated. Test completion criteria are established with the persons responsible for the project and the reporting system agreed on.

4. The testers create concrete test cases during the implementation phase. Test cases for use as part of smoke or regression tests will be automated in accordance with the degree of automation that has been determined. The test environment as well as any test data that are required are organised. If requested, the testers assist the development team in the creation of automated unit tests with the aim of uncovering as many defects as possible during the development phase. The prepared test cases are performed after the requirements have been finalised. Smoke tests serve to review the general testability of a new version. Automated regression tests serve to increase test coverage and free up time for testers so that they can perform more creative manual tests. Confirmation tests check whether debugging has been successful.

## 5.3 Test Levels at a Glance

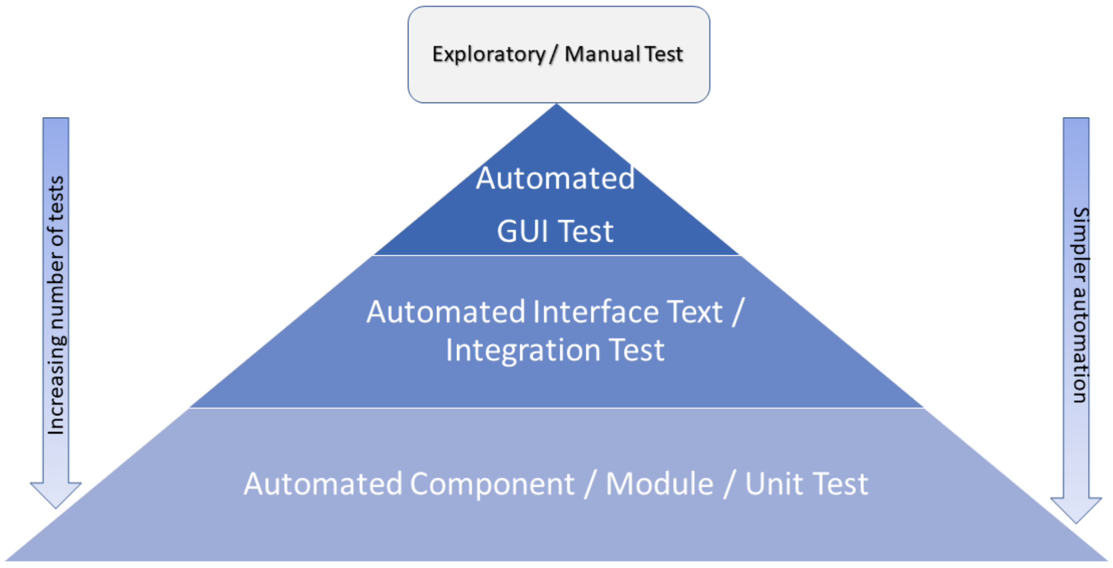
The following table describes the individual test levels within the Austrian Post in greater detail, stipulates the responsible organisational unit in each case and defines the requirements for the particular test environment:

| Test Level | Contents | Responsible Unit | Test Environment |
|---|---|---|---|
| Component or integration test (= unit test – see below) | Test of the implemented classes, components and interfaces against the specifications contained in the design and use of unit testing tools. Performed on an automated basis during check-in or build. | Software Development | Development environment, daily or ongoing build. |

| Test Level | Contents | Responsible Unit | Test Environment |
|---|---|---|---|
| System test | Test of the (sub-) system against functional and non-functional requirements; focus on the perspective of the user and suitability for purpose; use of black-box process, non-functional tests, and test automation. | Scrum teams | Own test environment. If required, multiple test environments. Deployment as required or by arrangement. |
| System integration test | Tests the integration of the system into the operating environment; interoperability with other systems; focus on interfaces, operational and business processes.<br><br>After the successful automation of the individual systems, their dependencies or end-to-end processes are to be tested on an automated basis. The architecture of the key words created in the system is carried out with a focus on the system integration test to guarantee problem-free reuse and simpler further development. | Scrum teams | Dedicated integration test environment/acceptance environment. |
| Acceptance test | Formal test of the fulfilment of acceptance criteria; basis for acceptance by business division/product owner. | | Acceptance test environment. |

While test levels build upon one another in a traditional approach, they are performed simultaneously in an Agile approach. These levels define different test types which entail both specific testing methods and tools at the same time.

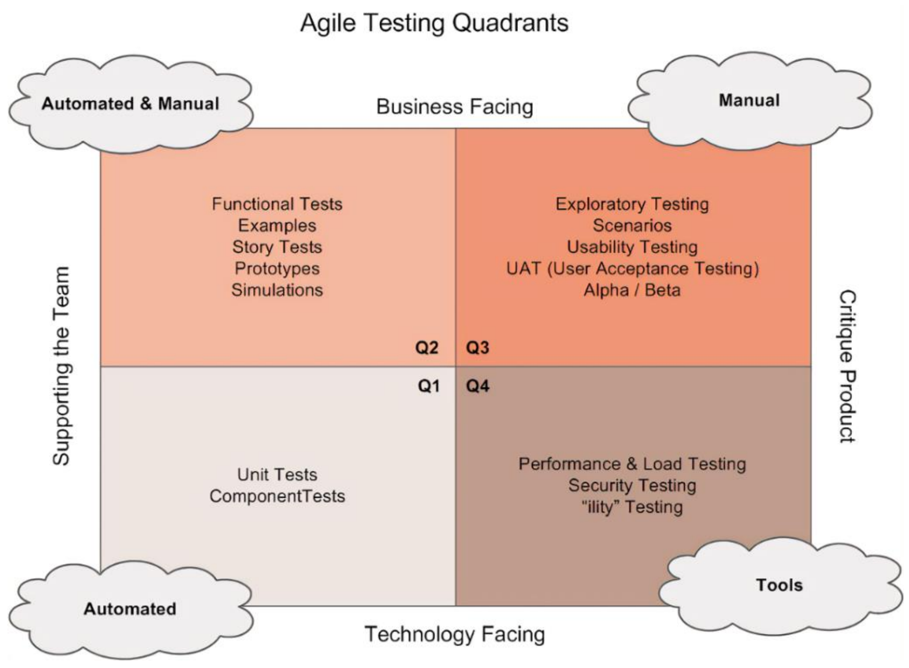
The test pyramid is intended to help clarify the right combination of different tests. This needs to be determined during the development of a test strategy. Ideally, a large proportion of quick and easy-to-maintain **unit tests** will form a broad and solid foundation for the pyramid. Integration tests form the middle level of the test pyramid. They normally have longer implementation times and are more complex to maintain. For this reason, they should be used to review critical interfaces in a targeted manner. The top of the pyramid consists of the comparatively slow and sometimes maintenance-intensive **UI/system tests**. These provide a good insight into whether the application functions as a whole. However, they are completely unsuited to testing all of the possible branches in the respective code. Consequently, their number should be kept to a minimum and as many tests as possible shifted to the lower levels of the pyramid.


Exploratory / Manual Test
Automated GUI Test
Automated Interface Text / Integration Test
Automated Component / Module / Unit Test
Increasing number of tests
Simpler automation

## 5.4 Test Types at a Glance

In addition to the various testing levels, this handbook also differentiates between basic test types. Test type is understood to mean interrelated test activities that are focused on a single test objective (usually a quality characteristic).

It is also necessary in an Agile project to keep an eye on all of the quality factors during sprints at all times. The four Agile test quadrants provide information on this:



The first quadrant contains tests which support teams and are technical in nature, meaning they are typically performed by developers.

The second quadrant shows tests which are performed by the team to ensure that functions have been implemented as they were planned. Automation is also used for these tests, though to a much more limited extent. Acceptance test-driven development and business-driven development are often used as methods.

The third quadrant contains tests which scrutinise the product's suitability for business use. These tests look at the software from the perspective of the end user very intensively. They are performed manually.

The tests in the fourth quadrant are specialised tests which often require experts with specialist testing knowledge (e.g., security testing) or tools (e.g., load and performance testing).

### 5.4.1 Smoke Tests

This kind of test is normally performed in a relatively short period of time ("timeboxed") and checks basic functionalities. The aim is to ensure a minimum level of quality before the test item enters the following test phase. Smoke tests must always be performed in the "target test environment" and are the responsibility of the test team.

Ideally, smoke tests should be performed in the presence of developers (or a representative from the development team) so that blocking defects can be jointly analysed and rectified as quickly as possible.

### 5.4.2 Functionality Tests

Business functions are the use cases or logical processes of a software solution. Business functionality testing checks the completeness and accuracy of the functions provided by the software. A piece of software's business functionality is complete when all of the functions required of it can be performed (i.e., all use cases with every possible outcome and all logical branches). Business functionality is correct when the actual result matches the desired result for every test case.

### 5.4.3 Non-Functional Tests

The technical attributes of a software solution correspond with its non-functional requirements.

#### 5.4.3.1 Reliability Test

The reliability of a piece of software is defined by its ability to maintain a predefined level of performance under specified conditions. During performance tests, special attention is given to maturity (avoidance of crashes), error tolerance (where incorrect data has been input) and the recoverability of the software in the event of a crash.

- Maturity testing is carried out indirectly when the software is subjected to a functionality test.
- Testing of error tolerance (or robustness) is performed by confronting the software with unexpected input values, either via the user interface or other interfaces.
- Recoverability testing is performed by inducing a system outage and testing the software's recovery routine.

#### 5.4.3.2 Penetration Tests (Security Tests)

The security of a piece of software is defined by its ability to protect information and data from being accessed (and misused) by unauthorised persons. The security level for each piece of software that is developed must be determined together with Post IT's Chief Information Security Officer (CISO). Depending on the security level that has been set, explicit security tests (primarily penetration tests) must be scheduled. Security tests are undertaken as part of existing framework agreements with security test providers. The CISO handles these assignments.

#### 5.4.3.3 Load and Performance Tests

System performance is defined as the sum of several individual characteristics such as response time, processing time and capacity. The measurement of these characteristics is the goal of load and performance testing:

- The response time is the time interval between the point in time at which an operation is entered and the point in time at which the result is output.
- The processing time is the time interval between the start of a background process and its end.
- Capacity is the number of operations (and/or the number of parallel users) that the system can process simultaneously.
- System utilisation is the percentage of system resources used by this system on average.

All of these characteristics can be recorded in absolute numbers as actual measured values and compared with target values.

#### 5.4.3.4 Usability Tests

The usability of a piece software is defined by its ability to be understood by, learned by and attractive to its users. Assessing the usability of the system is the goal of usability testing. Usability is seen in relation to users' expectations and any compliance requirements which may apply, meaning it is difficult to measure. In usability testing, an expectation profile must be agreed on with users and then combined with standards. The criteria which the particular system has to fulfil are derived from this process. Usability tests are supported by user observation, usability lab or 'Thinking Aloud' methods.

#### 5.4.3.5 Transferability Tests

Transferability is understood to mean the ability of software to be transferred from one environment to another. This includes, among other things:

- Installation Tests: A test of a piece of software's installation routines (including a review of the release notes and installation instructions).
- Upgrade Tests: A test to see whether the installed version of a piece of software can be successfully replaced by a newer version.
- Coexistence Tests: A test of coexistence with other systems in a joint environment (with jointly used resources).
- Migration Tests: A check of the completeness and accuracy of data migration routes (a comparative test of the content of the old and new system, a comparative test of quantity, usage test).

#### 5.4.3.6 Compatibility Tests

The integrability of software is an aspect of its functionality and determines its ability to interact with other defined systems. The most important test in this regard is the interface test. The interface test is usually carried out across multiple levels:

- During development: Review of interface definition, review of the implemented interface against the interface definition.
- During the system test: Technical review of the interface for syntactical accuracy (data formats, attribute sequence, field lengths, content mapping) as well as a functional usage test in the course of validating business processes (focus on interface processes).

#### 5.4.3.7 Maintainability and Reusability Tests

Dynamic testing cannot be used to determine whether the requirements of maintainability, integrability and reusability have been fulfilled. The extent to which they have been fulfilled can only be ascertained through a static analysis of the design documents and the source code. These can be partly checked and measured using automated processes and subsequently evaluated by an expert.

Adherence to specifications regarding principles and patterns is reviewed with the aid of tools (SonarQube). The rules, their classification and 'Quality Gates' are described in the "Software Quality Guidelines" document.

### 5.4.4 Regression Tests

A regression test is understood to mean the repetition of test cases which have already been performed after any changes to the system have been made, e.g., during iterative development. The goal is to ensure that adjustments, enhancements and corrections to the system and data that have been made subsequently do not have any negative effects on functionalities which have already been tested and which produced positive results.

A degree of regression of at least 30% must be provided for. This means that 30% of the test cases which have already been created and implemented are to be repeated per iteration.

### 5.4.5 Confirmation Tests

A confirmation test is understood to mean the retesting of failed test cases after any errors have been rectified. It (only) serves as evidence that a defect has been successfully rectified. A confirmation test must be combined with a regression test in the rectified defect's functional environment.

### 5.4.6 Story Tests

Story tests are tests from a business perspective which are used to verify the delivered software as part of the implemented user story. When the story is planned, the team creates several story tests which cover the story's acceptance criteria. They are a combination of tests for the traceability of requirements and, on the other hand, necessary regression tests which result from this.

### 5.4.7 Exploratory Tests

Exploratory testing is independent of test case creation. Exploratory testing can be carried out at very short notice since test design and test implementation take place simultaneously and time does not need to be invested in the creation of test cases.

Furthermore, costs for test case creation and maintenance can often be eliminated. This is a considerable but often neglected cost factor.

- Test design and test implementation take place simultaneously. There are no clearly distinguished phases for test case creation and test case implementation.
- Test cases do not have to be created (although they may be).
- The emphasis is on learning. On the assumption that not all of the facts and required information are known at the beginning of a test, it is also necessary for the tester to engage in a learning process during test implementation.

### 5.4.8 Acceptance Tests

With respect to organisation, acceptance tests belong as part of user stories and the acceptance criteria defined within them. An acceptance criterion reveals whether the story is finished from a customer or user perspective. The acceptance test describes the steps which are to be carried out to determine whether the functionality in the sense of the acceptance criterion can be regarded as correct. Since acceptance tests should actually be carried out by customers (or at least in their presence), they are located both at the beginning (definition) and the end (performance) of a user story's development.

### 5.4.9 Stress Tests

A stress test involves raising the simulated load above the level expected during normal operation to the point at which functional errors occur or the response behaviour of the tested system transgresses defined limits. A continuously increasing number of users is often simulated for this purpose. Stress tests are used, for example, to ascertain system behaviour during and after overload situations, to determine the maximally acceptable load or to find weak spots in performance (bottlenecks).

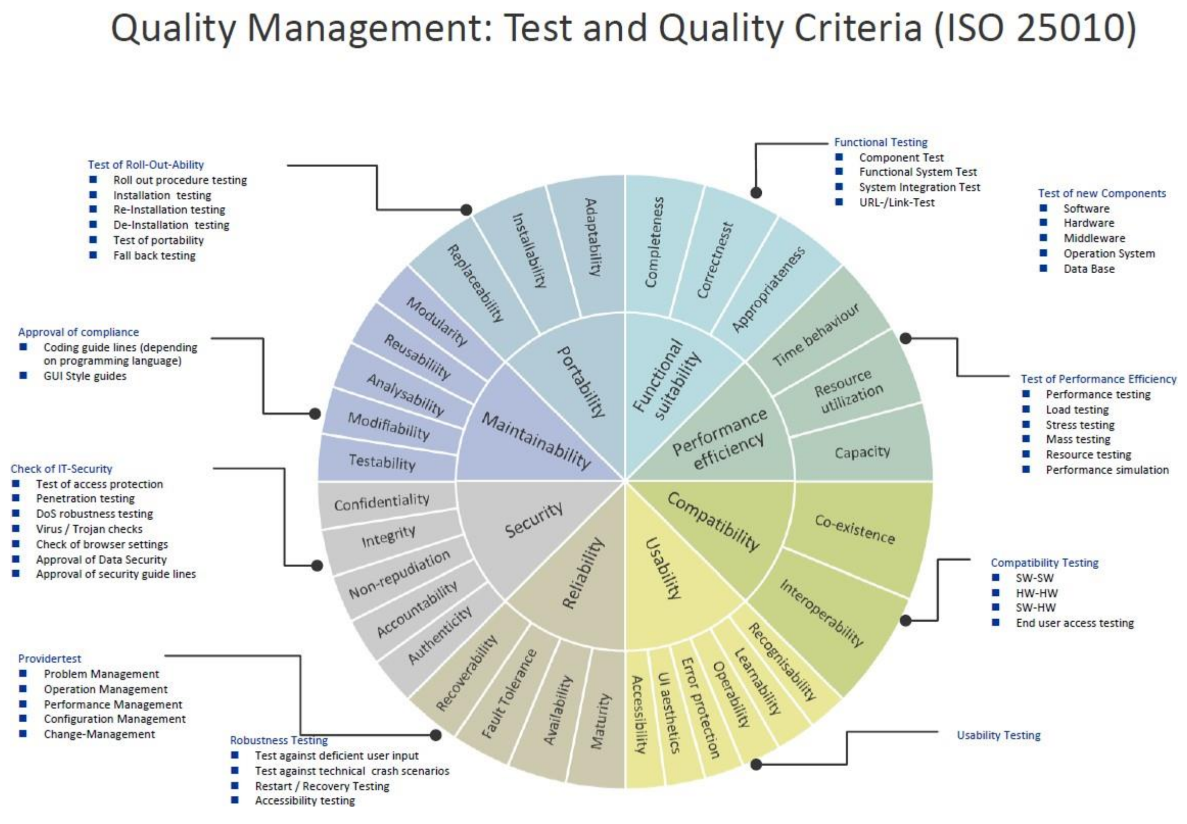
# 6 Test Planning

The goal of test planning is the “qualification” (what, why, how, when, who) and “quantification” (how much and at what price) of the testing for a project.

## 6.1 Risk-Based Testing

Tests must be designed in a risk-based manner to ensure that they contribute as much value as possible. (In this regard, see also “The Value Contribution of Testing” in the Testing Guidelines).

The software quality characteristics in ISO 25010 help with this step:



## 6.2 Planning for Traditional and Bimodal Performance

The test manager creates a test concept at the beginning of a project. To this end, the test manager has to fill out the *Test_Concept_Template*.

The standards contained in the Testing Handbook apply as part of this and do not need to be described in the test concept a second time. The test concept should only contain the planning information required for the respective project. Specifically, this includes:

- A description of the (product) risks.
- A test strategy aimed at minimising risk.
- Test items and performance characteristics (including delimitation).
- Acceptance criteria.
- Work packages, responsibilities, and time planning.

## 6.3 Planning for Agile Performance

During the course of iterative development, the team carries out testing activities together during planning and their daily meetings.

The following tasks are to be fulfilled on the basis of the content that has been determined for the sprints:

- The content for testing activities which are to be carried out is planned.
- Test depth is determined by reference to the criticality of requirements.
- A rough schedule is planned in consideration of the handover of test versions from development.
- Overall planning is optimised for deadline risks:
    - Handover times for test versions are brought forward.
    - Requirements are bundled.
    - The test depth for non-critical requirements is reduced.
    - Requirements are moved into the next cycle.
- The cycle planning is published within the project team.

## 6.4 Integration Tests – Planning

Features which extend across teams or systems must also be comprehensively tested by means of an integration test.

The following procedure must be followed in order to plan for this accordingly:

- Before Program Increment (PI) planning:
    - Cross-team or cross-system features are identified during ‘Business Refinement’ at the latest and marked with the tag “Integration”.
    - Cross-train integration tests are to be coordinated with the respective train and features/user stories are to be created in the corresponding backlog so that they can be presented and scheduled during Business Refinement or PI planning.
- During PI planning:
    - Features for integration tests (“Integration” tag) are identified in the course of dependency analysis using, for example, a Kendis board.

    - For tagged features, a user story is created for each team involved in the same sprint, even if no implementation is planned but test support (e.g., the creation of test data) is required.
    - User stories are scheduled synchronously as part of the same sprint for joint testing in accordance with the implementation dependencies of all of the teams involved.
    - User story titles: „IntTest | <Title>“ can optionally be marked with the tag “Integration”.
- After PI planning:
    - The technical details of features and user stories relevant for integration tests are analysed during refinement(s) and sprint planning(s) with an emphasis on testability.
    - The Test Manager in charge of the project is responsible for the organisational and coordinative activities as part of integration tests.
    - The costs for user story integration tests are estimated for each team separately. This can already be done for the first sprint during PI planning (partly lived practice).
- In the event of the scope being changed:
    - If there are changes to the scope (e.g., planned features/user stories will not be implemented or will be implemented later; unplanned features are accepted into the PI), the respective Test Manger(s), RTE(s), PO(s) and SCM(s) are to be informed.

# 7 Analysis and Design of Test Cases

## 7.1 Identification of Test Items

The basic prerequisite for the traceability and analysability of testing activities is a structured and systematically documented requirements analysis. Relevant test basis documents derived from the test objectives are analysed as part of this process and the requirements' structure derived from them. Relevant test basis documents include, among others:

- Specifications.
- Use case descriptions.
- Epics, features, and user stories in the case of Agile development.
- Business process descriptions.
- Interface descriptions.
- Software architecture model.

Since comprehensive test coverage for all requirements is not economically feasible, it is necessary to evaluate requirements with reference to the following criteria:

- Meaning for the user.
- Frequency of use.
- Criticality for use (risk).
- Acceptance criteria must be covered.

Automation goals as well as the degree of automation (the proportion of user stories with at least one automated test out of all of the user stories marked “automatable”) and rate of automation (the proportion of automated tests out of all of the tests) must be defined.
For example:

- CRUD (Create, Update, Delete) operations must be tested on an automated basis.
- Percentage staggered according to the priority of the requirements.
- Mixture of the first two variants.
- Previous experience from test automation projects which have already been completed.
- Specialist know-how about the test items.

## 7.2 Definition of Logical/Creation of Concrete Test Cases

In a further step, test cases are defined and assigned to each requirement. This process draws on equivalence class formation methods and threshold analysis methods or the different kinds of combinatorial analysis. The test cases defined in this way are also referred to as *logical test cases* (e.g.*,* the logging of a *consignment* in a *city*).

In a further step, logical test cases are recorded with concrete values (e.g., a letter to 8010 Graz). The scope of test case description is the responsibility of the Software Test Specialist and is dependent, among other things, on the prioritisation of the related requirement, on the extent to which the test cases are reused by third parties, and on regulatory requirements. Since the Austrian Post usually operates software solutions longer term, corresponding reusability as part of maintenance and further development is to be taken as a starting point. Any regression tests must be added

The following information from each test case *must* be documented irrespective of the required coverage:

- Unique test case name.
- Summary of the test case.
- Priority of the test case.
- Reference to the underlying requirement (reference to the test model).
- Prerequisites for test case performance.
- Post-conditions for test case performance.
- Necessary test data.
- Description of the test’s steps including any necessary input values.
- Expected result.
- Test case status (ready, reviewed, etc.).

## 7.3 Review of Test Design and Test Cases

If required, the test design (in the case of high-priority, technical or functionally complex requirements) should be reviewed and adjusted by third parties (for example, by an SA or PO).

The specific test case design for test automation can be found in the Test Case Design for Automation chapter.

# 8 Implementation and Performance

## 8.1 Procedure for Introducing Test Automation

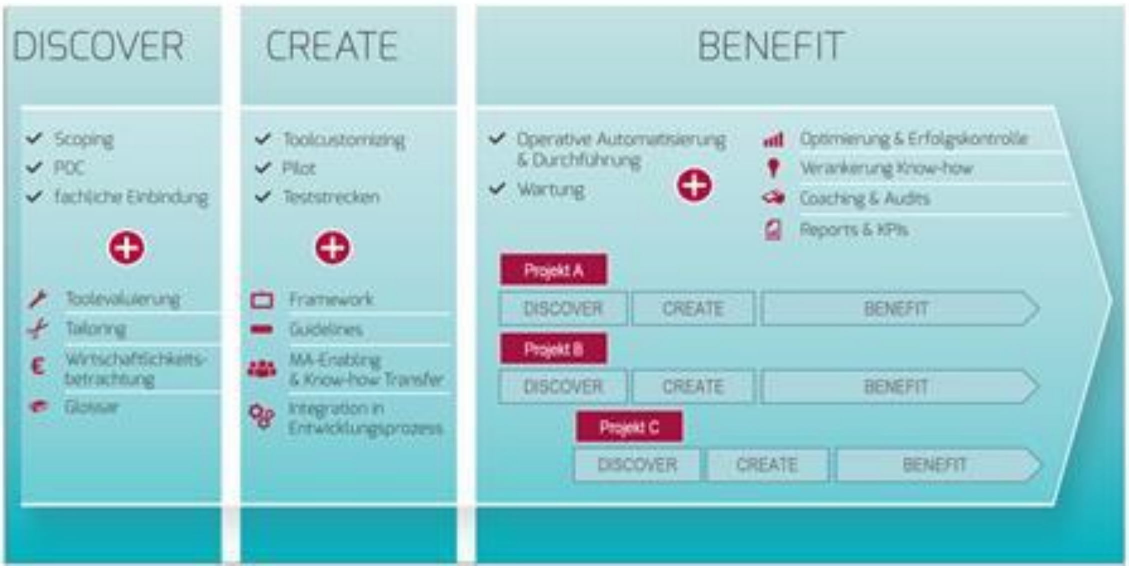
Test automation at the Austrian Post takes a holistic view of test automation activities in three phases:

During the “**Discover**” phase, requirements analyses, technical feasibility studies and, above all, an assessment of the economic viability of the intended activities are carried out.

The “**Create**” phase includes test automation activities that are already active in the form of pilot projects, the development of customer-specific test automation frameworks and early integration into the development process. Furthermore, work begins on building up know-how and enabling employees. To that end, guidelines, documentation, and a ‘Definition of Done’ for test automation code are created.

The “**Benefit**” involves intensive work on the test automation solution. In the process, particular attention is paid to the correct use of test automation patterns. Furthermore, permanent success monitoring and optimisation analyses are carried out.

In each project, work is once again carried out according to the three phases of the ‘Advanced Automation Approach’ in order to guarantee the ideal integration of project specific test automation into the overall picture and the automation framework.



This ensures its overall benefit across the entire system landscape and the ideal benefit of test automation in each project.

## 8.2 Preparation of the Test Environment

To avoid impairing testing activities through parallel development activities (and vice versa), it is necessary to prepare or provide a separate test environment for higher test levels (from and including the system test).

New test environments are only created where no test environment already exists for the item that is to be tested. The team always uses any pre-existing environments that are available (i.e., it does not build any “mirror systems”).

## 8.3 Preparation of Test Data

Test data can:

1. be gathered synthetically during testing;
2. transferred in an anonymous form from a data copy from the production system;
3. be retrieved by importing a data backup from the test system;
4. created synthetically using SQL scripts; or
5. be created synthetically using test data generators (specialised tools).

**The following applies to test automation:**
In general, synthetic test data are used for all test automation activities. In a server-client application structure, the following variants are used:

- In principle, web service interfaces are used.
- If not all of the required test data can be created via web services, creation will be carried out directly via the database layer. However, care must be taken to ensure a clean data structure in this case.
- If it is not possible to create data using web services or a database connection, automated creation can be carried out with the user interface in extraordinary circumstances.
- Alternatively, a test data generator can be used.

## 8.4 Test Entry Criteria

The following entry criteria are relevant to the start of the system test:

- Relevant and coordinated requirements.
- Unit tests have been carried out successfully during development.
- The software has been installed in the test environment.
- All of the (third-party) systems or corresponding testing aids (stubs and test drivers) required to perform the test are available.
- It was possible to carry out the smoke test successfully.
- Review of non-functional criteria using, for example, SonarQube (see the “Software Quality Guidelines”).

## 8.5 Test Performance

Every test performance must be logged. To this end, the following must be logged for every test case:

- The date and time the test was performed.
- The tester who performed the test.
- The result of the test (the status specified by the testing tool).

## 8.6 Automation Processes & Activities

The following chapter describes the underlying process as well as the activities involved in test automation. The processes run at exactly the same time as the defined test phases.

### 8.6.1 Determination of Requirements

- Input: Architecture concept, project test concept, Testing Handbook
- Activities: Conceptual workshop
- Output: Project automation concept

### 8.6.2 Test Case Automation

- Input: Logical test cases with status “Automation planned”, existing manual regression test cases
- Activities: Implementation of keywords, interface connections and test cases, development of framework components
- Output: Test automation code in the source code manager, linking with the functional test case, scheduling as part of the automated test run, keyword repository

### 8.6.3 Test Run Analysis & Reporting

- Input: The results of an automated test run (passed, failed)
- Activities: Classification of the “failed” test cases, filtering and evaluation of the results, logging of error tickets, correction, and adjustment of the test cases
- Output: Error description, ‘to dos’ for maintenance and further development activities, results report to the project team, classification of failed test cases

### 8.6.4 Test Case Maintenance and Further Development

- Input: Functional/technical amendments, results from the test run analysis and reporting
- Activities: Implementation and adjustment of the test cases, impact analysis
- Output: Amended artifacts (test automation code, test data, framework), impact analysis

## 8.7 Interruption and Resumption of Tests

All testing activities as part of system testing and system integration testing activities (in all work packages) will be interrupted if:

- the test system, the test environment or the required test items are not available or not available in their entirety (this may concern both the hardware and the software);
- the test item is highly flawed, and a test is severely hindered or is not of meaningful use;
- the poor performance of the system or individual test items severely impairs a test or means that it is not of any meaningful use;
- despite several test and confirmation test cycles, the system does not stabilise;
- owing to crashes, migration jobs provide data which is incomplete or which is demonstrably erroneous;
- not all of the (third-party) systems required for the test are available;
- comprehensive amendments to or expansions of the system which were signalled become necessary.

In all of these cases, if the test needs to be interrupted, a deviation (Class 1: Critical or Class 2: Serious) is logged.

Corrective measures must be identified in coordination with the developers and with the highest priority level (resolution within a maximum of 24 hours after the error was logged).

The test is resumed after the successful implementation of the corrective measures has been checked (e.g., installation of a hotfix).

# 9 Test Automation

In general, test automation is mandatory as part of all projects. The test automation strategy can vary from project to project. In any case, the following points are to be observed in the process:

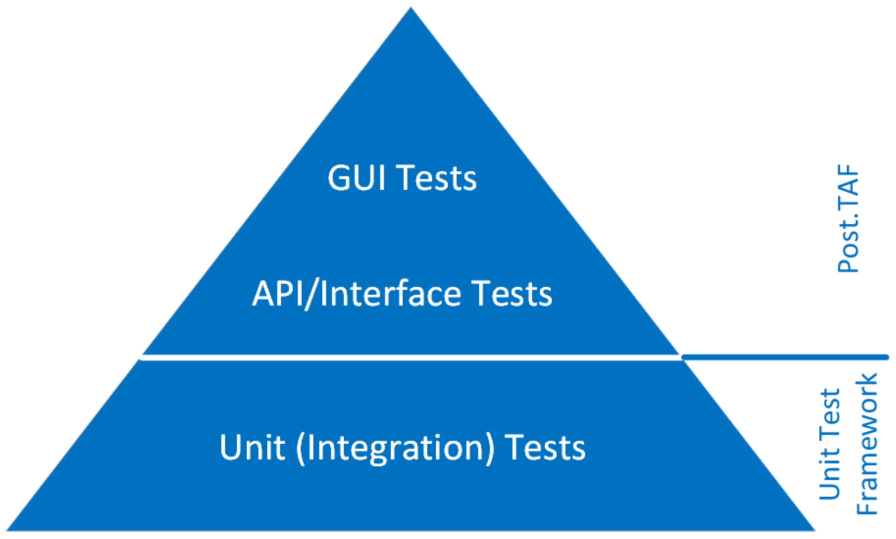
- Test automation is to be set up along the entire length of the test levels.
- Automated tests must be created in accordance with test case design (exception: unit tests).
- A link between automated test cases (exception: unit tests) and test cases in Azure DevOps must be established.
- It must be possible to visualise test automation in Azure DevOps (widget on the dashboard).
- Automated regression test suites must run on a nightly basis at the very least.
- The Austrian Post’s tooling is to be used for automation.
- Test automation must be maintained (including after the project has ended).

Any deviations must be agreed on with Test Management in advance.

## 9.1 Test Automation Along the Test Levels

Test automation includes tests at the unit level and at the API and GUI level. The following diagram illustrates the different test levels and the tools which are to be used.

Unit tests comprise the base, which is implemented during development, while API/interface tests and GUI tests are carried out within the test automation framework (or Post.TAF).



## 9.2 Test Automation in the Stages

The different test levels are carried out at different stages of development:

- Unit (integration) tests: DEV Stage
- API/Interface tests: TEST Stage
- GUI Tests: TEST Stage
- Integrative tests: (typically) ACCEPTANCE stage*

*This is usually because system integrations already exist at the TEST stage, but all relevant systems are typically only integrated at the ACCEPTANCE stage.

## 9.3 Test Case Design for Automation

In test case design, the maintainability and reusability of test automation solutions is considered at an early stage through the use of test automation patterns. The following patterns are implemented:

- Keyword driven: A keyword refers to the grouping of several functional actions into a single method. Keywords can be accumulated and then used in test cases on a discretionary basis. In this way, a kind of tool kit is built up which can be used in the creation of test cases depending on the particular functional requirements.

  Data-driven: The data-driven approach aims to conduct several data combinations using the same test case. In this way, it is not necessary to create a test case for each individual data combination. Instead, the test data are fed into the test case from an external source. One test is performed per test data configuration.

- Arrange, act, assert: The arrange-act-assert pattern provides a way of structuring test cases. In this regard, “arrange” refers to all of the necessary preparatory steps that are required for the actual test performance. “Act”, in turn, refers to the actual action and actual test, while “assert” refers to the checking or, so to speak, verification of the test objective. Segmenting a test case in accordance with these three steps results in simplified readability and strengthens the recognition value of test cases within the project’s code.
- Page object: A page object is the abstraction of a coherent part of the application under test. In it, all of the relevant interaction possibilities with this part of the application are depicted as methods. As a result, a page object can include a collection of keywords for a particular part of an application.
- Data provider: A data provider organises the provision of the test data required for a test case or an entire test run. Given that test data should not be managed in the test case directly, a data provider which takes over this function is implemented.
- Client: A client is the actual implementation of the interaction with the test item. As part of this process, the actions of the above-mentioned page object are implemented with the particular automation tool. In the case of the Austrian Post, this is a Selenium and Appium client, diverse web service clients (REST, SOAP) and a WinAppDriver client. The implementation of the interaction as a client makes it easier to swap automation tools without losing the entire test automation code.

The use of patterns for the creation of test automation solutions helps to increase maintainability, reusability, and further development.

It is recommended that the checklist set out in the appendix be used for the selection of test cases for automation.

## 9.4 Multilayer Test Automation

A fundamental element of the chosen automation approach is automation on different application layers (multilayer). The following chapter describes the multilayer test automation approach.

Testing on multiple application layers facilitates the creation of efficient and compact test cases since not all tests need to be performed on the GUI layer and, as a result, all of the application’s layers. The credo “Test functionality at the place where it is found” is particularly important in this regard.

As a consequence, the test cases which have been created are easier to maintain and contribute to a superior test coverage of the individual layers.

The following layers can be tested within a typical client web server architecture:

### 9.4.1 GUI Layer

User interactions should be tested on the GUI layer. In particular, the display of business-critical areas and functions must be covered by the GUI layer. User and role concepts that control display and input modalities are also an important component in GUI layer tests.

#### 9.4.2 Web Service Layer

A review of pure server functionality that can be accessed via web services. This is to be tested on the web service layer directly. This is recommended for testing combinations and calculation logic in particular. Since the trend is to move application logic from the front end into web services, it is also possible to perform automated work-flow tests via web services. A web service client is used in the Post.TAF for this purpose.

#### 9.4.3 Database Layer

Tests on the database layer are primarily used for the comprehensive review of data which have been entered and which cannot be fully displayed on other layers. Furthermore, tests on the database layer should also ensure the integrity of new or migrated data sets.

Another important use for the tests which have been created is the preparation and reworking of other test cases. At the web service and database level in particular, data for other test cases can be rapidly and efficiently prepared as well as removed again after the test has been performed. This applies to test cases on the GUI layer in particular, since the preparation and post-processing of the required data purely over the GUI layer is very slow and, above all, prone to error.

## 9.5 The Austrian Post Test Automation Framework – Post.TAF

In general, the Post.TAF including integration into Azure DevOps (linking to test cases, nightly test runs, visualisation of results) is to be used for test automation in all projects.

Multilayer test automation can be performed with the aid of the Post.TAF. The following clients are also available for the individual layers:

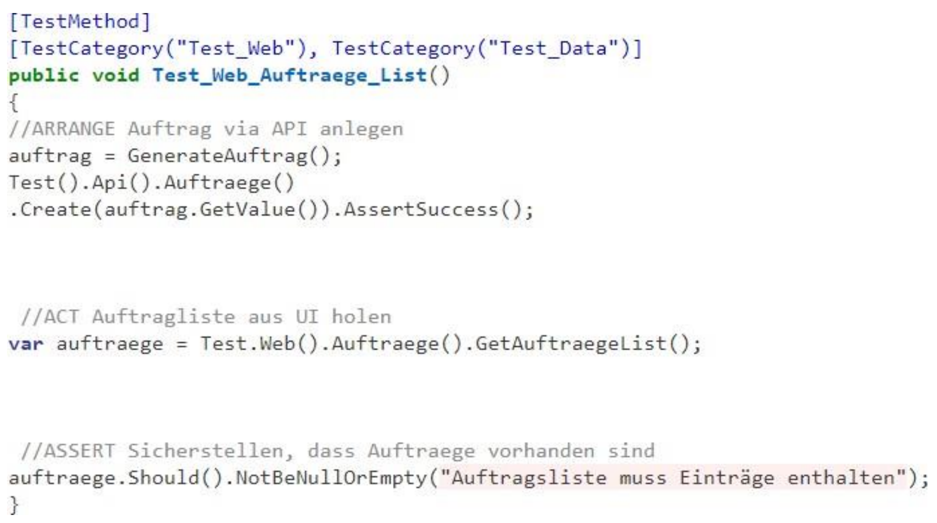
- GUI Layer:
    - Selenium (web).
    - Appium (mobile).
- Web service:
    - REST.
    - Soap.
- Database:
    - No explicit client, but possible.

The Post.TAF as well as the individual clients can be downloaded from the Austrian Post's feed via NuGet packages. At the same time, each individual test solution and, as a result, all of the keywords, data providers etc. contained therein can be made available as a NuGet package.

A combination of tests from different layers is possible in the Post.TAF and should be given preference for the generation of test data, for example. For instance: A user can be created in the same test case using a web service client and subsequently used on the GUI layer for testing.

## 9.6 Example of an Automated Test Case

An example of the Arrange-Act-Assert principle by reference to an exemplary and straightforward test case in the Post.TAF. In it, a task is generated using the REST client before tasks are brought into the frontend using GUI clients:



```
[TestMethod]
[TestCategory("Test_Web"), TestCategory("Test_Data")]
public void Test_Web_Auftraege_List()
{
//ARRANGE Auftrag via API anlegen
auftrag = GenerateAuftrag();
Test().Api().Auftraege()
.Create(auftrag.GetValue()).AssertSuccess();



 //ACT Auftragliste aus UI holen
var auftraege = Test.Web().Auftraege().GetAuftraegeList();



 //ASSERT Sicherstellen, dass Auftraege vorhanden sind
auftraege.Should().NotBeNullOrEmpty("Auftragsliste muss Einträge enthalten");
}
```

## 9.7 Infrastructure for Test Automation

In addition to the test environment of the particular system under test, test automation requires an agent which can perform automated tests. In general, this agent is installed on a virtual machine (VM) in Azure and added to the general test centre agent pool in Azure DevOps.

Besides the installation of the agent, the VM must also have the particular software required for the performance of the test (e.g., dotnet core SDK, Selenium Driver, Browser, etc.). At the same time, all of the agents in the pool should operate the same software stack to be able to conduct every possible test run.

## 9.8 Inclusion in the SDLC

Test automation is to be integrated into the CI/CD pipeline of the respective project (e.g., for smoke test sets which run at every check-in). Alternatively, a separate build/release pipeline (e.g., for nightly test runs) needs to be set up.

## 9.9 KPIs

In general, the test automation status is visualised in Azure DevOps (via the Test Results widget on ‘Dashboards’) or in SonarQube in the case of unit tests.

The progress of test automation is gauged through the use of the “degree of automation” and “rate of automation” KPIs.

The degree of automation identifies the number of user stories with at least one automated test out of all of the user stories marked as “automatable”.

The rate of automation measures the number of automated tests out of all of the (manual + automated) tests.

Both KPIs can vary with respect to target values depending on the particular project. The following target values apply: 50% of the automatable user stories must have at least one automated test case. Furthermore, 30% of all test cases must be automated.

The progress of unit tests is measured using code coverage in SonarQube. A target value of 80% applies to code coverage. Likewise, a code coverage level of 80% is required for legacy systems in relation to new code and, concomitantly, coverage must not sink in the course of time.

## 9.10 Test Automation for Integration Tests

Test automation should also be used for integration tests. In this regard, it is necessary to consider which test cases that have already been created are suitable for this purpose or whether a separate integration test suite should be created. When doing so, it should be noted that these tests - in contrast to user-story-based test automation - must run at the acceptance stage to ensure that all relevant systems are actually integrated. When creating one's own integration tests, automation should be implemented at the feature level based on risk.

# 10 Test Data Concept

## 10.1 Procedure

During the procedure, a distinction must be made depending on the test environment:

- **Acceptance Systems:**
  In principle, acceptance systems fall under the responsibility of the accepting party (business division). In this case, representative data configurations are taken out of production and loaded into the respective test system prior to acceptance or testing activities. For cross-system end-to-end tests, data are created and used in all of the affected systems directly. There are no systematic activities for the purpose of data generation across the entire system landscape.
- **Development/Test Systems:**
  In this case, representative data configurations are created either manually or on an automated basis in the respective systems by means of tool support (e.g., test automation, dedicated individual solutions[1]). In some cases, the basic dataset is created from an anonymised copy of the acceptance test.

### 10.1.1 Test Data in Test Automation

In the area of test automation, the test data which are required are created automatically in the course of the performance of test cases in the respective systems and removed again if necessary. These test data are completely synthetic and are generated precisely for the test objective.

[1] During test automation, test data are frequently created during the 'Arrange' step (see "Generation of Test Data") and individual solutions should be understood to mean tools that can only be used for specific projects currently. There is, for example, a test data generator in the Austrian Post's full data collection system (abbreviated to VDE in German).

As a result, an automated test case has a completely unique dataset for each performance which behaves correctly during functional testing.

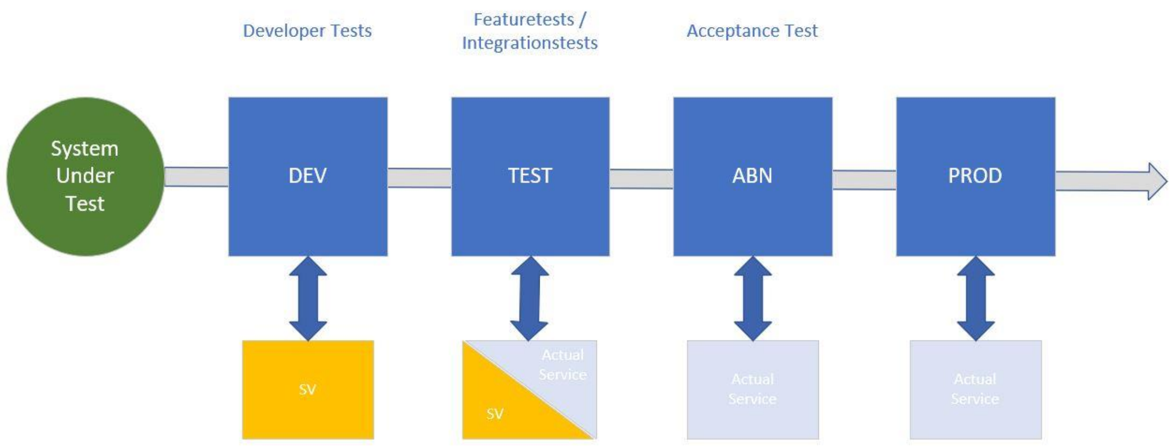
#### 10.1.2 Test Data in Test Environments

Test data are made available via different production methods in different test environments. These methods are described in greater detail in the “Types of Test Data Production” section.

### 10.2 Definition and Goals

Test data are data which have been created synthetically or which have been extracted from the production environment and anonymised. They are explicitly used for software tests. The aim is to provide test data required for systematic testing in a high-quality form, anonymised or preferably synthetic on demand for all relevant stakeholders.

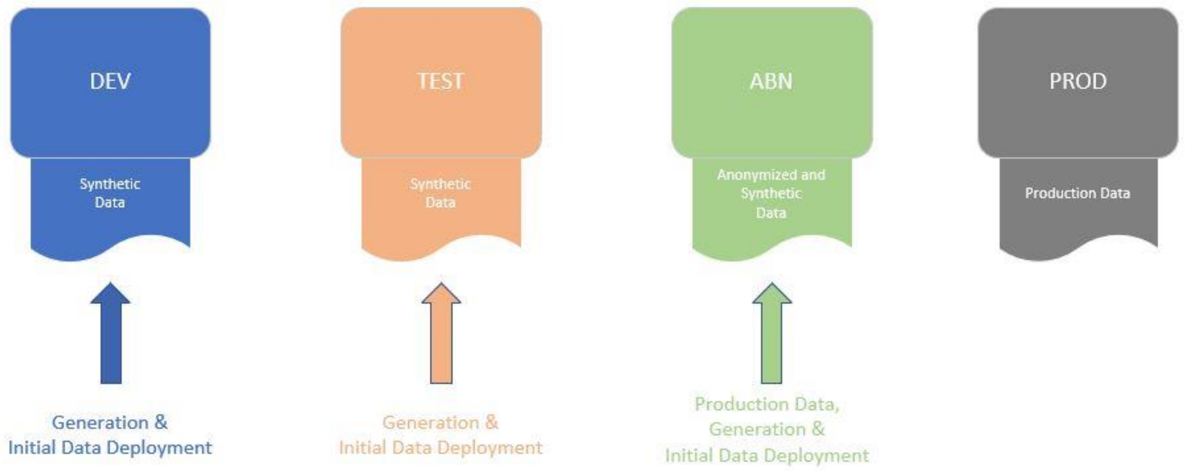
Furthermore, it must be possible to provide valid test data across system boundaries to facilitate integrative testing. In the process, service virtualisation is used in development and test environments to minimise dependencies on external services and their test data.



The use of this procedure means a state is reached whereby only synthetic data are still used during the DEV and TEST stages.

### 10.3 Types of Test Data Production

This figure illustrates the intended test data types and generation methods depending on the particular environment:


DEV
Synthetic Data
TEST
Synthetic Data
ABN
Anonymized and Synthetic Data
PROD
Production Data
Generation & Initial Data Deployment
Generation & Initial Data Deployment
Production Data, Generation & Initial Data Deployment

The production of test data can be carried out in different ways:

1. Generation: Test data are generated synthetically using the respective system under test (SUT) or by means of tool support.
2. Initial generation using database deployment: During the deployment of database solutions, certain data are automatically created in the database which has been set up (e.g., by means of SQL script) depending on the test environment in which deployment is carried out.
3. Extraction of production data: Test data are generated and anonymised through the extraction of test data from production.
4. Hybrid approach: The hybrid approach pursues a combination of the aforementioned approaches in order to produce the required data.

Please note that generation (bullet one) and production using deployment (bullet two) are always to be preferred to the other approaches. The extraction of data from production should be seen as a last resort and avoided as far as possible.

### 10.3.1 The Generation of Test Data

The production of a specific test dataset is carried out as part of manual testing using the SUT or a test data generator. In the process, the desired data are specified and created in the test system accordingly by way of processing in the SUT or in the generator.

In projects in which test automation is used, the generation of valid test data is applied within automated test cases. In the process, the automated test case creates synthetically generated test data itself. These are then used during the test run and cleaned up automatically at the end of the test. As a result, the use of production data within test environments is not required.

The logic with which test automation creates data needs to be defined once at the outset. In the process, an analysis of what constitutes a valid dataset in the system and how it can be created in a technically correct manner must be performed. If the analysis has been carried out and the logic defined on a one-time basis accordingly, this logic can also be used for other test projects via an artifact (e.g., a NuGet package).

There are several tools available for the generation of test data, especially for manual testers. *ELFRIEDE* was created to generate Austrian Post users and related shipments. *VDE Testcenter* was created for VDE's webservice interfaces. The *Test Data Generator* can be used to create barcodes for IRIS tests. An overview of and contact persons for the three tools is available on the Group-IT Wiki.

If data cannot be created on an automated basis, greater testing costs are to be expected as data will have to be created manually and possibly by different persons (e.g., in the event of cross-system data creation).

#### 10.3.1.1 Initial Creation Using Database Deployment

A defined test data inventory can already be imported during the deployment of the database solution. In the process, data are entered into the database directly through the execution of scripts in the course of the deployment pipeline and are available for use after deployment has been completed.

It should be noted with this type of generation that test data are consumed by testing activities and can only be reset to the defined inventory through a new deployment.

Initial creation is particularly useful in the area of master data, i.e., data that are required for correct functionality so as to create a foundation for further activities in the respective test environment. In principle, these master data are static and are not usually manipulated by test operation, meaning initial creation during deployment is sufficient to enable test operation.

Another option is to fill the database with mass data through deployment. This makes it possible to recreate production-related conditions with respect to data volumes.

#### 10.3.1.2 Test Data from Production Extracts

This option should be avoided as far as possible. If used, it should only be applied to acceptance systems. For example, test data are extracted from production systems using a backup-restore-mechanism and then loaded into test environments. In the process, the type of data should be considered, and an assessment made as to whether data protection measures need to be implemented in order to ensure the test system is GDPR compliant. Test data of a personal nature will be anonymised and pseudonymised accordingly.

## 10.4 Management and Provision of Test Data

The management of test data is based on the management and versioning of artifacts. One simple possibility is the use of internal database projects which are checked into the source control management system. These guarantee consistent versioning and make processing and traceability easier.

Another important activity as part of the management of test data is permanent quality management. Any amendments to data models are also considered in the test data in order to keep them up to date.

The updating and versioning of test data facilitate the timely creation of defined states and reduce sources of error in the event of a roll-back.

The provision of test data is carried out using a process which has been defined by the team. The process will consist of different activities depending on the system and the stage. As part of this, the automated preparation of the target systems and, after successful import, initial automated checks of the test data are also performed in addition to the actual import of the test data.

Furthermore, all directly and indirectly affected persons are to be informed prior to the provision of new test data.

## 10.5 Anonymisation and Pseudonymisation

Anonymisation and pseudonymisation are measures taken where personal production data are used as test data.

Strictly speaking, the term anonymisation means that certain data (e.g., sensitive personal data) are deleted.

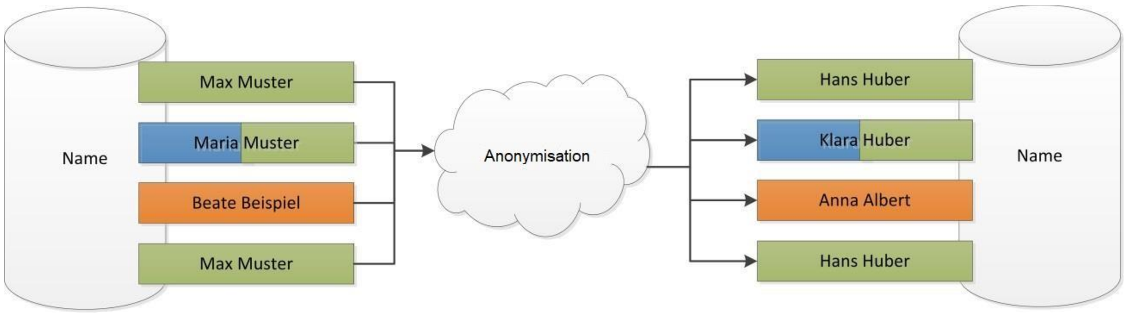
The term pseudonymisation means that certain data are replaced with other plausible data, a process which make the identification of specific individuals impossible or, at the least, extremely difficult. However, the difference between them is hardly ever observed in practical language use and the two techniques must be used at the same time in order to protect personal and sensitive data. For that reason, the two terms are used synonymously in this concept, with the exception of detailed specifications where it genuinely makes sense to differentiate between the two.

General technical considerations with respect to the manipulation of data constitute a prerequisite for all of the following approaches. Such considerations result in part from the complexity of the systems and functional dependencies.

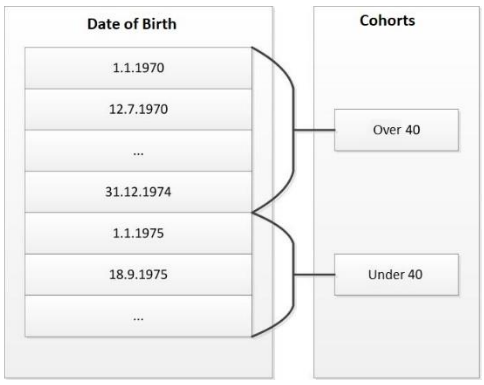
### 10.5.1 The Same-In-Same-Out Principle

This principle can be applied to addresses and names, for example. The same input values always get the same output values.

If it is not possible to reconstruct the input value from the output value, this is known as a one-way hash. This prevents real data from being reconstructed from anonymised output values where a reasonable level of effort is expended.



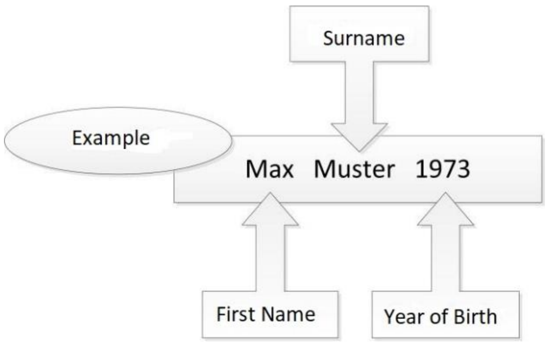
### 10.5.2 Cohort Building



This approach is important for dates of birth and address spaces. This creates general groups which limit an otherwise free characteristic, such as date of birth, to a fixed value range if the data are manipulated. For example, this approach can be used to create functionally meaningful cohorts of persons above and below a specific age. As a result, manipulation of the date of birth is only permissible within this range, meaning the year of birth can only be changed to a limited extent.

Cohorts can be defined broadly or narrowly in accordance with functional requirements. This can also be done iteratively if it turns out that a very broad cohort needs to be more narrowly defined subsequently in order to deliver better results.

### 10.5.3 Splitting Up Concatenated Entries



It can be the case that several elements, each of which carries specific information, are connected to one another within various data fields. For example, an address field can contain both the street and postcode. Since the data each carry different logical information, which is to say, they each need to be anonymised in a different way, they must be split up prior to their being manipulated so that they can subsequently be dealt with separately.

The three methods which have just been described serve to clarify possible ways of anonymising data. If it is necessary to anonymise data, this should be done in accordance with one of these three methods.

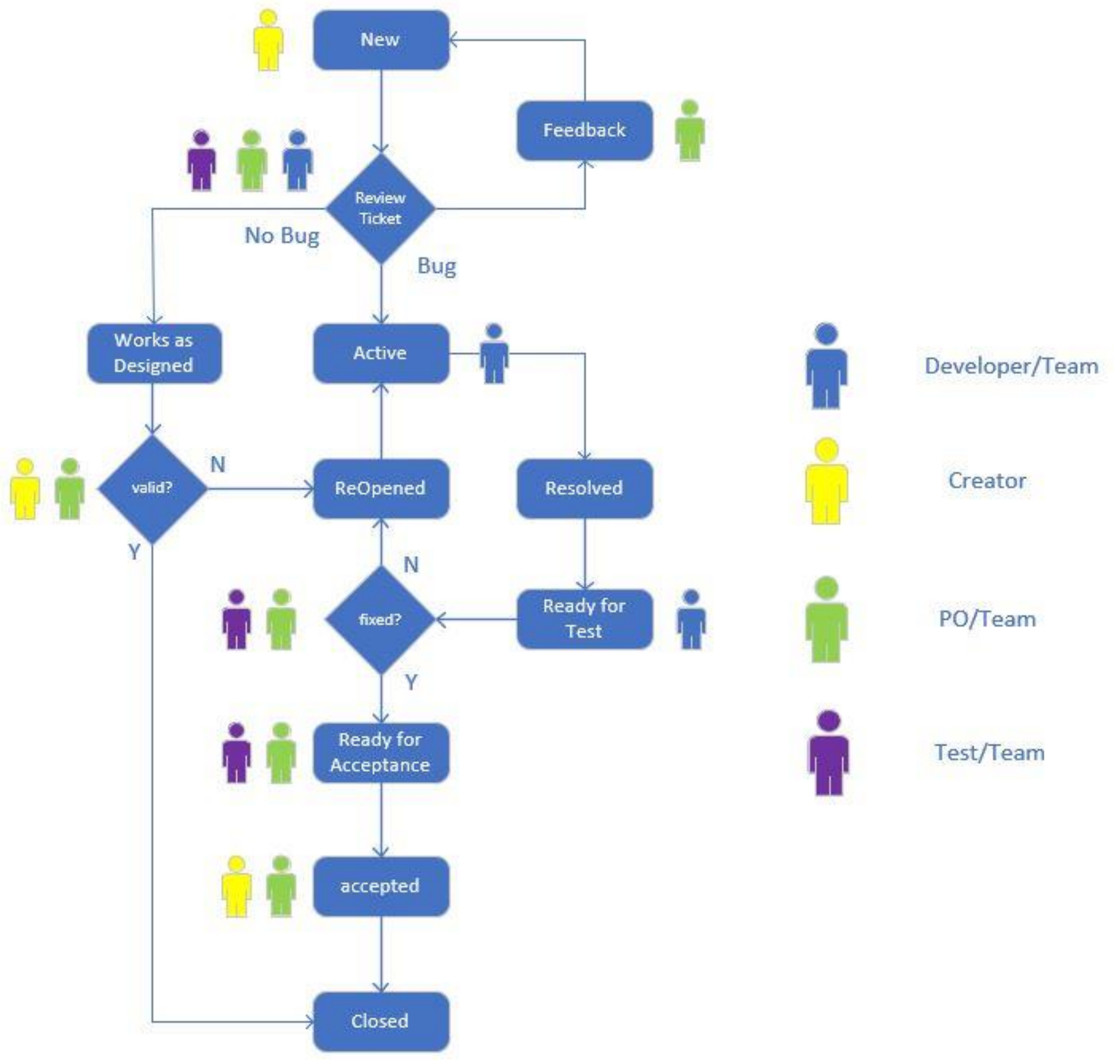
# 11 Deviation Management

## 11.1 Goals

All deviations which occur after the development phase of a user story (from “Ready for Acceptance” on) must be recorded as the “Bug” work item type in the deviation management tool and described in sufficient detail. The defined deviation management process guarantees the traceability of deviations which have been discovered as well as the debugging process. A responsible person is given charge of a deviation at all times, making it impossible for any defects to “get lost in the system”. Beyond that, the deviation management process also provides the basis for evaluating test progress and the quality of the product with respect to scope, status, and trend.

## 11.2 The Lifecycle of a Deviation

The management of the lifecycle of a deviation is based on the following workflow:



If defects have been remedied and are ready for testing, the following tests must be performed:

- A confirmation test to check whether the defect has in fact been remedied.
- A regression test to ensure that no new defects have arisen as a result of the debugging process.

## 11.3 Description of Deviations

The creator must record the following information for each deviation at the least:

- Title.
- Area path.
- Iteration path.
- Date and time at which the defect was observed.
- Notifying party.

- Environment/system/tenant
- Information about the test case which was performed/reproduction steps
- Severity of the deviation (to be set manually)
- Description of the deviation (including the required behaviour that is expected)

## 11.4 Categorisation of Deviations

Deviations are categorised according to the following levels of severity (classification of the impact of the deviation on the user during the test phase and when in operation)[2]. Every defect must be allocated to one of the following categories when recorded:

| Classification | Meaning |
|---|---|
| **Urgent or Very High** | The proper use of a significant part of the system is not possible or limited to an unreasonable extent. Very serious consequences for business transactions and/or security result from the defect.<br>**Example:** System outage without restart, data loss or destruction, incorrect results during the time-sensitive mass processing of data. |
| **Severe or High** | The proper use of a significant part of the system is severely restricted. The defect impacts on business transactions and/or security.<br>**Example**: Incorrect or inconsistent processing which subsequently leads to restrictions during ongoing operations; however, has no long-term consequences (e.g., batch data cleansing possible). |
| **Medium** | The proper use of a significant part of the system is slightly restricted.<br>The defect does not have a significant impact on business transactions and/or security.<br>**Example**: The user receives logically incorrect notifications (e.g., error notifications). |
| **Low** | The proper use of the system is possible without restriction.<br>**Example**: The graphic representation of content needs improving. |

## 11.5 Deviations in Test Automation

Unlike defect management as part of manual testing, an analysis of the previous test run must be performed in the case of test automation. As part of this, unsuccessful test cases are assessed according to the following criteria:

- Defect.
- Environmental problem.
- Automation error.

2 Defect categorisation is based on IEEE-Standard 1044. The classes formulated in the standard, *urgent, high, medium, low, none,* were translated into the particular area of application in order to facilitate a practice-orientated classification.

- Test data error

Should the analysis deliver the result “Defect”, the same defect management process as for manual testing is subsequently initiated.

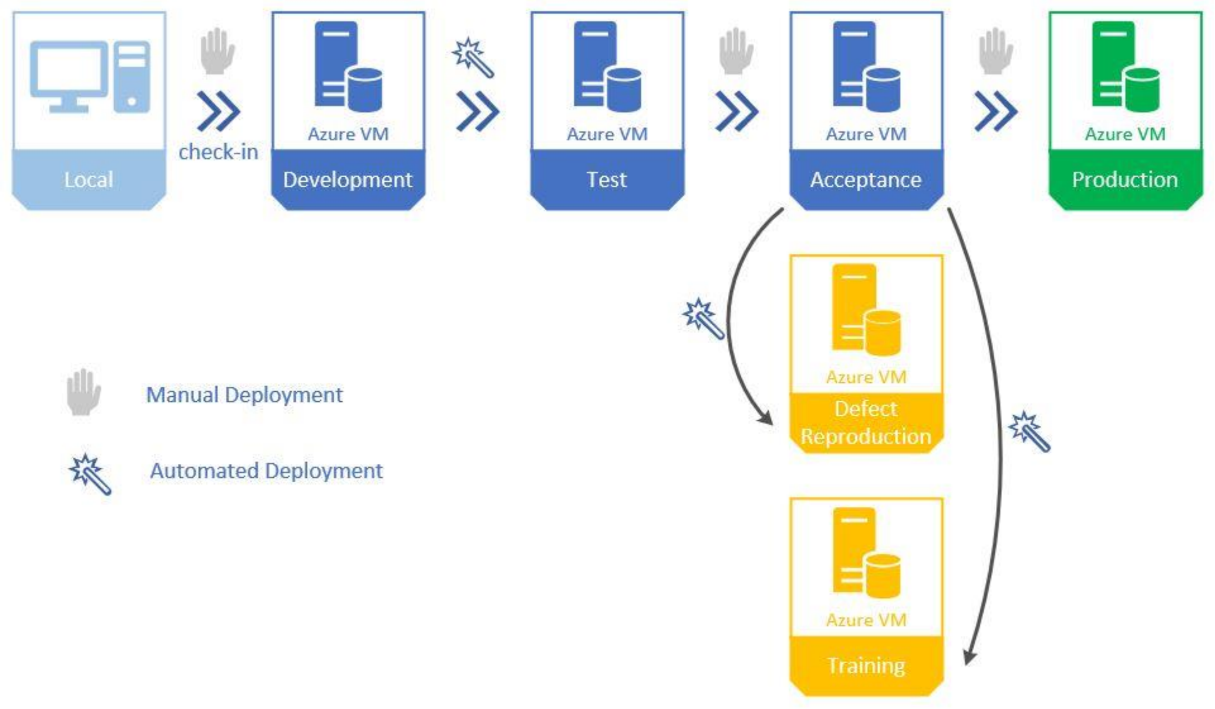
The test automation specialist is required to revise the test case or the test data for all other criteria.

# 12 Test Environment

## 12.1 Staging Concept

Every software development project requires several test environments for a seamless development and testing process. On the one hand, the different environments for development and testing should support the lifecycle of an application. On the other, the integration of different systems should be testable as early as possible.

The following staging structure is the comprehensive form of the process. The environments showed in it are described in greater detail below:



### 12.1.1 Local

| | |
|---|---|
| Areas | Development |
| Responsible person/unit | Individual developer |
| Task | Integrated test environment based on the same VM images as for subsequent test levels for prompt debugging |
| Azure integration | Copy of the cloud image for local performance |
| Deployment to next stage | The developer decides him-/herself whether his/her code should be checked in the SCM |

### 12.1.2 Development

| | |
|---|---|
| Areas | Development, testing |
| Responsible person/unit | Development |
| Task | Automated pre-testing of potential release candidates and troubleshooting |
| Azure integration | DevTest-Lab Environment - Development |
| Deployment to next stage | As soon as the automated tests have been successfully concluded, deployment to the next stage will take place automatically. |

### 12.1.3 Testing

| | |
|---|---|
| Areas | Testing |
| Responsible person | Testing |
| Task | Automated and manual tests |
| Azure integration | DevTest-Lab Environment - Testing |
| Deployment to next stage | Depending on the test objective, all manual and automated tests must run faultlessly. Deployment to the next stage only takes place by way of manual intervention. |

### 12.1.4 Acceptance

| | |
|---|---|
| Areas | Business division, operations |
| Responsible person | Business division |
| Task | Final test level for the business division, complete integration without virtualisation |
| Azure integration | DevTest-Lab Environment - Acceptance as well as business division's central hardware |
| Deployment to next stage | Only through manual intervention. |

### 12.1.5 Defect Reproduction

| | |
|---|---|
| Areas | Business division, development, operations |
| Responsible person | Operations |
| Task | Production-related environment for the purpose of reproducing defects from production |
| Azure integration | DevTest-Lab environment - Defect reproduction |

When a particular version is deployed in the production environment, it is also deployed in the defect reproduction environment at the same time. This environment is normally dormant, meaning the machines have not yet been started. The aim is to have a simple way of reproducing a defect promptly in the event that one occurs in the production system. The defect reproduction environment means this is also possible when other newer versions are already being tested on other systems.

#### 12.1.6 Training Environment

Away from the regular environments for the development and testing of various systems, a single environment dedicated to training should be available. This is an exact clone of the defect reproduction environment and is only activated for training purposes.

## 12.2 Staging Concept – Minimum Characteristics

It may be the case that not all test environments are built/required depending on the particular project. In such a case, minimum characteristics for the staging concept must still be put in place to be able to guarantee the smooth running of a project. The following stages are required as part of the minimum characteristics:

- Local.
- Development.
- Testing.
- Acceptance.
- Production.

The responsibilities and deployment procedures (prerequisites, manual or automated depending on the stage) remain unchanged.

## 12.3 Requirements for Test Environments

Requests for new test environments are made to Azure Operations. The following information is to be provided so that the test environment can be created quickly.

- Desired environments.
- Technical requirements for the particular environment (e.g.: OS, CPUs, RAM, hard drive size, etc.).
- Cost centre.
- Requester.
- Project name/code.
- Network configuration (public, internal).

The different processes involved in test environment management are always gone through where one or several test environments have been requested. These are described in greater detail in the following chapter.

## 12.4 Test Environment Management Processes

The following processes are used as part of test environment management at the Austrian Post.

### 12.4.1 Test Environment Requirements Analysis

The aim of this process is to collect, analyse and specify the requirements on the test environment (or on multiple test environments running in parallel) as well as the operational requirements for the test environment. In the process, a stable basis for the requirements placed on the test environment must be created. Furthermore, the criteria and conditions for the acceptance procedure and operation of the test environment are also developed as part of this process. Another topic for consideration is the specification of the requirements for test automation, which should also be carried out in the course of this process.

### 12.4.2 Design & Configuration of the Test Environment

The goal of this process is to create a design for and, after that, as detailed a description of the configuration as possible for the construction of the test environment. The description of the configuration must contain information which is as precise as possible with respect to the software, hardware, and network components for the construction of the test environment.

### 12.4.3 Construction of the Test Environment

The aim of this process is the construction of a test environment in accordance with the specific configuration. In the course of this process, the network, hardware, and software are set up, installed, and configured in their respective positions. At the same time, the required test drivers, mock items, and testing tools are also installed and configured where these have been specified. Moreover, user authorisations must be set up, assigned and managed. If possible, the test environment is integrated into the development team’s continuous deployment process so that it can be automatically recorded, configured, and anchored in the release chain.

### 12.4.4 Testing the Test Environment

The aim of this process is to check whether the requirements placed on the test environment have been fulfilled and whether the test environment is ready for use. This is done by carrying out the defined acceptance test cases (e.g., the testing of diverse workflows). The test results, defects, deficiencies, and the status of the test environment are logged and tracked. The final release notes for the test environment are documented and communicated.

### 12.4.5 Operation of the Test Environment

The aim of this process is to guarantee accuracy (in the sense of the specifications) and efficiency in the operation of the test environment and test items (products) for the duration of the test performance. In the course of this process, the test items are tested in the test environment.

#### 12.4.6 Support & Maintenance

The aim of this process is to ensure an acceptable level of test environment operation through maintenance and advice so that testers can carry out their testing activities effectively.

In principle, the project team that requested the environment has full administrative control of the development and test environment machines in order to make necessary amendments and carry out maintenance work. There is also the opportunity to obtain support from a test environment management employee. These support and maintenance tasks relate to the machines in the test environment and their readiness for use as set out in the specifications. Support of the test item being operated in the environment at the functional level cannot be guaranteed by test environment management. Advice, support, and activities in the area of staging, continuous integration and continuous deployment can also be carried out by Test Environment Management depending on the particular requirement.

## 12.5 Staging Documentation

The current documentation for the existing stages has been stored on the Austrian Post's Azure DevOps. The documentation can be accessed via the following link:

Post AG Staging Documentation

# 13 Evaluations

## 13.1 Status Information

During the performance of a test, updates on the test progress that has been achieved, error statistics, achievement of test completion criteria, performance indicators and any problems or impediments as part of the test are provided on a regular basis.

The following KPIs are made available:

- The number of defects found per requirement.
- Defect categories.
- Automation level: The number of test cases carried out on an automated basis is compared with the total number of test cases or the user story.
- In new projects, there is at least one test case for every acceptance criterion.
- Defect trend: Open defects (in development/in testing) are compared with closed defects.
- Test progress: Test status (implemented test cases versus open ones) of each requirement and the entire project.
- Test coverage: Test coverage corresponds to the coverage of all user stories without the "not test relevant" tag which have been given the status "Closed", "Accepted", "Ready for Acceptance", "Tested", "Ready for Test", "Resolved" or "Active". User stories that have been covered by test cases and which have a value of less than 90% are given the status "red". A value of less than 100% but greater than 90% is given the status "yellow" while a value of 100% is "green".

- Burndown chart: The progress of the user stories which have been developed and tested.

# 14 Specific Features of Acceptance Testing

## 14.1 Goals

The acceptance test serves as an organisational aid in a software development project and constitutes part of the formal acceptance procedure. It is intended to ensure that uncertainties on the part of the decision-maker in the business division are reduced as far as possible in the course of accepting the developed product. During the acceptance test, a check is made to see whether the software that has been commissioned fulfils all of the contractual criteria and, as a result, can be accepted by the business division. In the course of Agile development, the product owner accepts the user stories after completion as the business division's representative.

## 14.2 Responsibility

The client/business division is responsible for the organisation and implementation of the acceptance test. The test team can support the organisation and implementation of the acceptance test on request. This includes the organisation of the business division's testers, the creation of test cases and the construction and provision of the test environment.

## 14.3 Infrastructure

The client/customer is responsible for providing the workstations which are required, office space for the acceptance test and the availability of an acceptance test environment. The test team can help if required. The acceptance tests are to be carried out in a test environment which is independent of the development and test system.

The specific configuration of the system landscape as well as the definition of the acceptance test environment must be arranged with the client in good time.

## 14.4 Procedure

### 14.4.1 Prerequisites

The business division has provided employees with sufficient availability for the preparation and performance of the acceptance test. The acceptance criteria exist as part of the contract, as part of the specifications or as acceptance criteria in the user stories.

### 14.4.2 Preparation of Acceptance Test Cases

Taking the acceptance criteria as a basis, the acceptance team creates acceptance test criteria. The aim is to check every acceptance criterion with at least one test case. The aim of this process is the development of a binding acceptance test case catalogue. It must be possible for the acceptance testers to carry out and evaluate all of the test cases contained within this catalogue within the time frame that has been designated for acceptance testing.

The following criteria should play a role in the preparation of acceptance test cases. Test cases should:

- Bring the software's suitability for use into focus.
- Be written from the perspective of the user.

- Review the execution and validation of business processes.
- Process chains and lifecycles.
- Consider all of the (third party) systems as part of the test (i.e., observe process "end-to-end").
- Build on the system test's previously completed test level and thereby avoid all possible redundancies.

### 14.4.3 Entry Criteria

The following entry criteria for acceptance testing need to be fulfilled:

- The business division has defined the acceptance criteria in the contract, in the specifications and/or in the user story.
- Release has been granted from the previous test level of the system test (i.e., the system test's test completion criteria have been fulfilled).
- A separate test environment is available for the acceptance tests or organisational measures (code freeze) prevent changes from being made to the software in the test environment.
- It was possible to install the version released from the system test in the test environment successfully.
- All relevant (third party) systems are available for testing.
- A smoke test has been carried out successfully.

### 14.4.4 Performance of Acceptance Tests

After the entry criteria for the acceptance test have been verified, the acceptance testers carry out the test cases.

Care must be taken to ensure that each test performance is logged in a verifiable manner (responsible acceptance tester, date and time of performance, test environment, performance result) and that each defect which occurs is recorded.

In an Agile approach, the acceptance test is carried out incrementally, i.e., each system component which has been completed is presented to the business division. The final acceptance test is carried out with the final increment and, as a result, the submission of the entire system. Outstanding test cases and a regression test are carried out at this point. Integration tests are performed contemporaneously during sprints and are to be coordinated with dependent teams.

### 14.4.5 Test Completion Criteria

The following criteria are recommended for acceptance testing for the purpose of evaluating test completion by the client. These need to have been determined by the time implementation begins at the latest:

**With respect to test progress:**

| Weighting of the requirement | % Implemented test cases |
|---|---|
| Low | 70 |
| Medium | 85 |
| High | 100 |

**With respect to defect categories:**

| Defect categories | Unresolved defects |
|---|---|
| 1 Urgent/Very High | No unresolved defects |
| 2 Severe/High | No unresolved defects<br>(All defects rectified and verified through retesting) |
| 3 Light | Unresolved defects up to a maximum of 20%<br>(Not yet rectified or verified through retesting) |
| 4 Trivial | Unresolved defects up to a maximum of 40%<br>(Not yet rectified or verified through retesting) |

**With respect to acceptance criteria:**
When all of the agreed acceptance criteria have been demonstrably met (i.e., confirmed by the acceptance test) by the software.

### 14.4.6 Acceptance

The client can confirm acceptance on the basis of the achievement of the objectives (with respect to the test completion criteria as part of the acceptance test).

# 15 Testing Tools

## 15.1 Azure DevOps

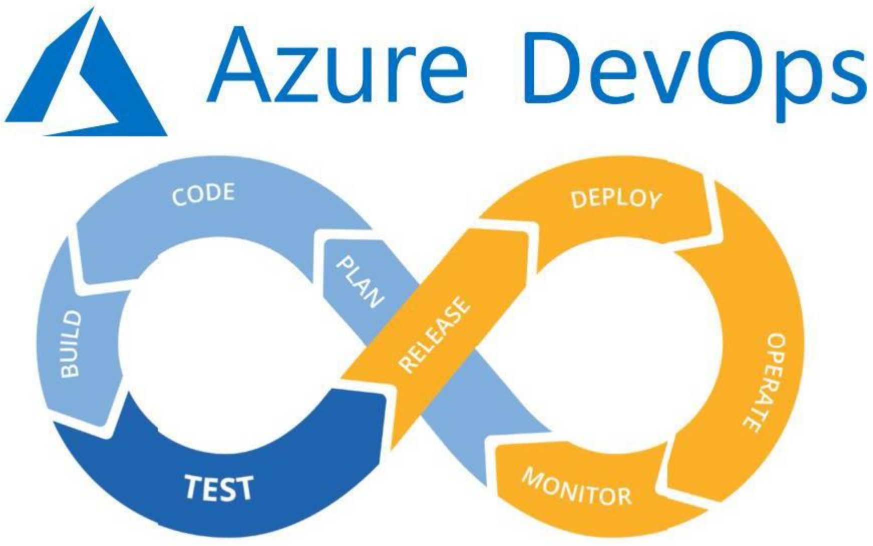


Figure 11 - Testing Tools Portfolio

Microsoft Azure DevOps (formerly Visual Studio Team Services) serves as the central interface for all activities within the software development lifecycle and, as a result, for testing or test automation. Among other things, Azure DevOps assists in the reporting of test results (dashboards), the maintenance of source code in automated test cases (code), the derivation of test cases from requirements (work), the management of test execution, (build and release), the documentation of manual and automated test cases (test) and the management of test clients (agent queues).

## 15.2 Azure DevOps – Test Tab

Scope of application: Test management, manual test performance and reporting.

The creation and management of a functional description for test cases which are to be automated is carried out under the 'Tests' tab in Azure DevOps. Furthermore, the 'Tests' tab in Azure DevOps serves to guarantee requirements traceability by linking between work items in the "Test Case" category and "User Story".

## 15.3 Azure DevOps – Pipeline Tab

Scope of application: Build and release including automatic test performance.

Azure DevOps is the development environment for the creation of automated test cases: All automation code is created and managed in Azure DevOps regardless of the GUI technology that is to be tested. Furthermore, all development activities for the purpose of creating test data providers and similar necessary modules for test automation are carried out with Azure DevOps. GIT is used for the purpose of version control management. The repositories for test automation can be found under the "Code" tab in Azure DevOps "Group-IT" team.

## 15.4 C#

Scope of application: Multilayer test automation.

The core of the test automation solution is built using .NET or C#. Automated test cases are developed in C# and carried out using unit test frameworks and Azure DevOps. The programming language connects clients from the different layers (Selenium, WinAppDriver, Appium client) and also makes its own clients available for the purpose of connecting web services and databases via frameworks such as RestSharp or Entity Framework.

## 15.5 WinApp Driver

Scope of application: UI Desktop application test automation.

WinApp Driver is used for the automation of desktop applications. In the process, all UI elements are identified programmatically. Interaction with all of the UI elements is carried out via programmatic actions, including application start and all the necessary clean-up work.

Documentation: https://github.com/microsoft/WinAppDriver

## 15.6 Selenium – WebDriver

Scope of application: UI Web application test automation.

The Selenium open-source framework is used for the automation of all web applications. In the process, the necessary Selenium – WebDriver components are integrated into the development environment using NuGet packages. Given that Selenium now offers several components for the purpose of web automation, it should be explicitly pointed out here that only Selenium's WebDriver components are used in the course of automation activities.

Documentation: https://www.selenium.dev/documentation/

## 15.7 Appium

Scope of application: UI Mobile application test automation.

The Appium open-source framework is used for the automation of all mobile (Android and iOS) applications. In the process, the necessary Appium WebDriver components are integrated into the development environment using NuGet packages.

Documentation: http://appium.io/docs/en/about-appium/api/

## 15.8 Smartbear ServiceV

Scope of application: Simulation of services.

The ServiceV virtualisation software from Smartbear is used for the purpose of reducing dependencies and to support faster development and testing.

## 15.9 Smartbear Cross-Browser Testing

Scope of application: Cross-browser testing for test automation and manual testing.

The "Cross-Browser Testing" cloud solution is used to test web services both manually and using automated processes in a way which is realistic, but which avoids incurring the high purchase costs associated with different end devices with different browsers, different operating systems etc.

## 15.10 Microsoft App Center

Scope of application: Device cloud for mobile test automation.

This is used for mobile solutions in test automation. Applications are tested here with the aid of a device cloud.

## 15.11 JMeter

Scope of application: Load and performance tests.

This is used when performing load tests on client-server applications. Jmeter can be used in very many different ways. These facilitate, among other things, the application and execution of existing TA tests via command line.

Documentation: https://jmeter.apache.org/usermanual/index.html

### 15.12 Xamarin

Scope of application: UI Mobile application test automation.

The Xamarin.UI test framework which uses NUnit is used for the automation of acceptance tests for mobile applications. The tests interact with the UI like a user: text entry, typing on buttons and gestures (e.g., swiping).

Documentation: https://docs.microsoft.com/en-us/appcenter/test-cloud/frameworks/uitest/

# 16 Roles and Responsibilities

## 16.1 Roles

### 16.1.1 Test Manager

The test manager is responsible for all testing activities with respect to projects, CRs and releases. The test manager's role is shown in greater detail in the RACI matrix below.

| **Responsible** | **Accountable** |
|---|---|
| Set-up of test dashboards | Test progress |
| Testing ambassador | Creation of platforms for knowledge exchange |
| Preparation of test reports | Test abort |
| Definition and follow-up of quality KPIs (with respect to test and code quality) | Escalations |
| Partner to Agile teams, team leaders and project leaders from a QA perspective | Test results |
| Coordination of cross-team and/or cross-system testing | Creation of a portfolio for the further development of QA |
| Coordination of standards for the use of Azure DevOps (tickets, test cases, test planning, etc.) | Participation in the planning and organisation of test environments |
| Checking of defect documentation and defect tracking | Resource/goal prioritisation |
| Test concept | Assessment of a given test's usefulness |
| Transparency to the end of quality (not only testing) | Remove any obstacles out of the way of testers |
| Further development of testing guidelines | Review of requirements/scrutiny of requirements with reference to testability |
| Guidance of decision-making with respect to testing tools | Introduction of state-of-the-art testing approaches |
| Reality check for testing guidelines, standards, and strategy | Risk management |
| Definition of, training in and checking of adherence to testing standards and procedures | Adherence to quality guidelines (code coverage, test automation coverage) |
| Creation of framework conditions for business division testing (if required) | Adherence to testing guidelines (manual and automated tests) |

| | |
|---|---|
| Test completion work (lessons learned) | |
| ‘Practice for Testing’ community | |
| Participation in the definition of the DoD and monitoring of adherence to the DoD | |

| **Consulted** | **Informed** |
|---|---|
| Writing of test case | Newly planned projects |
| Coordination with requirements engineering with respect to acceptance and test criteria | Project results (start, stop) |
| Test case management (definition, reviews) | Communication blockers and challenges as part of the project/testing |
| Project planning (schedule) | Scope |
| Continuing professional development of testers | Deployments |
| Test environment management in cooperation with test environment specialists | |
| Business division training sessions | |
| Coordination with respect to approaches | |
| Impact analysis of changes on testing | |
| Planning of test approaches for projects (methods) | |
| Performance and/or organisation of requirements reviews | |
| Development of the testing strategy | |
| Analysis of defects in production, reworking, etc. | |
| Test data management including cross-system data | |
| Identification of demand for testing resources | |

The test manager’s tasks do *not* include the following in particular:

- Participation in scrum teams.
- Creation of test cases (explicit test cases).
- Operations and/or roll-out planning.
- Staffing of scrum teams.
- Test performance.
- Performance and/or organisation of requirements reviews.
- Provision of booked items.
- Preparation of project status reports.
- Resolution of problems within the implementation team.
- Performance of system integration.

#### 16.1.2 Test Specialist

The test specialist (tester) facilitates testing activities in the project through the preparation and performance of test cases.

**Main tasks:**

- Creation of test scenarios, test cases and test data.
- Derivation of logical and specific test cases from data models, specifications (e.g., process models, use case models, etc.) and requirements using common testing methodologies.
- Performance of test cases manually and on an automated basis.
- Evaluation, documentation and tracking of defects and/or deviations.
- Documentation of test results.
- Documentation and tracking of defects.

#### 16.1.3 Test automation specialists

Automation experts implement test scripts for the test team and create automated test suites from them (e.g., for regression testing). Test automation specialists are also responsible for automated test data generation.

**Main tasks:**

- Conception, realisation, and further development of test automation solutions with the aid of suitable tools.
- Implementation of test scripts with the aid of automation tools.
- Support of testers in the performance of tests and coordination of test automation solutions.
- Performance and monitoring of automated test cases.
- Evaluation and documentation of test results.
- Continuous improvement and maintenance of test scripts.
- Construction and expansion of automated test frameworks.
- Responsibility for automated test data generation.
- Provision of advice and support to the test manager and specialists in the use of test automation (where should automation begin (GUI; services; backend); cost estimates; time planning; meaningful degree of coverage).

#### 16.1.4 DevOps Engineer (System Team)

The system team assists Agile teams in technical matters with respect to DevOps. These include, among other things:

- Test environment management (e.g., IaC - Infrastructure as Code, ARM templates, MS App Center).
- CI/CD Pipelining.
- Administration of diverse development tools (e.g., Azure DevOps, SonarQube).
- Set-up of service virtualisation.

It should be noted in this regard that the System Team acts as an enabler. The aim is for these teams to be able to perform these tasks independently.

## 16.2 Responsibilities

| Area | Responsible Person/Unit |
|---|---|
| Test environment | Test environment management (System Team) & Operations |
| Azure DevOps | System Team & Operations |
| Contents for tests | Test team |

| | |
|---|---|
| Creation of manual/automated test cases | Test specialist/test automation specialist |
| Performance of manual/automated test cases | Test specialist/test automation specialist |
| Analysis of automated test runs | Test automation specialists |
| Deviation management | Test automation specialist & test specialist |
| Requirements management | Team leader/product owner |
| Reporting | Test automation specialist & test manager/test specialist |