Service Validation and Testing

ITIL® 4 Practice Guide

AXELOS.com

22 November 2019

Contents

1 About this document

This document provides practical guidance for the service validation and testing practice. It is split into five main sections, covering:

- general information about the practice
- the practice's processes and activities and their roles in the service value chain
- the organizations and people involved in the practice
- the information and technology supporting the practice
- considerations for partners and suppliers for the practice.

1.1 ITIL® 4 QUALIFICATION SCHEME

Selected content from this document is examinable as a part of the following syllabuses:

- ITIL Specialist Create, Deliver and Support
- ITIL Specialist High Velocity IT.

Please refer to the respective syllabus documents for details.

2 General information

2.1 PURPOSE AND DESCRIPTION

Key message

The purpose of the service validation and testing practice is to ensure that new or changed products and services meet defined requirements. The definition of service value is based on input from customers, business objectives, and regulatory requirements and is documented as part of the design and transition value chain activity. These inputs are used to establish measurable quality and performance indicators that support the definition of assurance criteria and testing requirements.

The service validation and testing practice involves reducing the risks and uncertainties that new or changed products and services introduce to the live environment. The practice does this by planning and performing appropriate tests.

The larger and more complex a system is, the more testing is required. However, exhaustive testing, even of smaller, simple systems, is typically impossible due to time and cost constraints. Therefore, choosing what to test is important. The key considerations when defining the scope and level of validation and testing are the:

- agreed requirements that a product or service must meet
- impact and likelihood of deviations from the agreed requirements.

Understanding the requirements in the context of the likelihood and impact of deviations facilitates an informed perspective of the important areas to test.

This practice is about being confident in the quality of service being tested. This is not the same as saying that it is flawless. Confidence is earned through testing in order to demonstrate that the service will perform as required, meets the requirements, and has no significant defects.

2.1.1 Service validation

Service validation is performed in the earlier stages of the product and service lifecycle (ideation and design). It is focused on confirming that the proposed service design meets agreed service requirements and on establishing acceptance criteria for the next stages (development, deployment, and release). These criteria will then be verified by testing the product and service components, products, and services.

Validation follows the structure of service requirements and usually covers utility, warranty, experience, manageability, and compliance. Other requirements may also be included.

Service validation ensures the definition, verification, and documentation of service acceptance criteria and informs the scope and focus of testing activities.

2.1.2 Testing

Based on the criteria identified through service validation, test strategies and test plans are developed and implemented.

A test strategy defines an overall approach to testing. Test strategies can apply to environments, platforms, sets of services, or individual products or services. The product and service lifecycle stages that are covered by testing may differ between products and services developed within the organization and those obtained from a supplier.

The service validation and testing practice has been greatly impacted by changes in the architecture management, software development and management, project management, and infrastructure and platform management practices. Agile methods, the digitization of IT infrastructure, service-oriented architecture, and the automation of software development and management have introduced new challenges and opportunities to the service validation and testing practice. To meet today's requirements, service validation and testing should be faster, more flexible, and continually evolving. This is only possible if the practice is closely integrated with the practices mentioned above and others, including the release management, deployment management, incident, and problem management practices.

Effective validation and testing are based on close collaboration between testers, developers, and operations teams, alongside enhanced tooling and automation approaches.

Another important trend is expanding validation and testing beyond the technical aspects of products and services to include user experience and perception.

Traditionally, service testing was the act of confirming expectations relating to explicit requirements by checking the expectations on how the software should or should not work, based on prior knowledge that is defined through requirement specifications. Today, testing also involves exploring and uncovering information about unexpected things, such as product risks and variables regarding:

- software
- ideas for software solutions
- artefacts created from the ideas
- user experience and user interface designs
- models and wireframes
- architecture and code designs
- code
- tooling
- processes.

2.2 TERMS AND CONCEPTS

2.2.1 Risk-based testing

Risk-based testing is a common term within the testing industry. Typically, people understand risk-based testing to mean testing (particularly explorative testing) that is structured and driven by different types of product risks relating to the features and product components that are being tested.

The focusing on risk is beneficial because it highlights how the service might fail. This can then be investigated to uncover information about the software and its quality.

Commonly within software testing, people focus on types of testing. Examples of types of testing include functional, regression, performance, security, usability, cross-browser, accessibility, end to end, and integration testing. These types of testing focus on different types of risk. For example, functional testing focuses on functional risks and regression testing focuses on the risks of the software regressing.

Although they tend to consider ten to fifteen types of testing, many teams only include between five and eight types of testing in their test strategies. Because of this, and because there are many types of product risks affecting services that are rarely associated with a type of testing, a focus on risk-based testing is important.

2.2.1.1 Discovering Risks

Service validation and testing practice activities that identify product risks are as important and valuable as activities that confirm that risks have been effectively addressed.

Service validation and testing activities that are conducted in the early stages of the product lifecycle output information about product risks, variables, unknowns, and so on. Contrastingly, testing activities that are conducted in the later stages of the product lifecycle uncover problems and other information about the actuals of the service, to which the organization can then respond. Even when services are operational, organizations should continue to uncover information about risks, variables, and unknowns. That feedback continues, but stems longer feedback loops back into the ideas, user stories, and designs.

For example, in software development, it is extremely rare for agile user story artefacts and acceptance criteria artefacts to focus on product risks. The text within these artefacts usually relates to general expectations regarding functionality or the interconnectivity of the software. It is important to identify risks relating to the user story as acceptance criteria are being defined.

After identification, risks should be captured. Mind maps are a common tool for this because they create a risk map that is accessible, lightweight, readable, and ready to be used throughout the product lifecycle service design activities and explorative testing at the later stages.

Identifying different kinds of product risks can be difficult, but there are ways to structure acceptance criteria and testing activities that improve the chances of success, such as:

- Consider the object of testing on a holistic level, then granularly, including the tangible and intangible artefacts. Actively consider the product’s, service’s, or component’s:
  - potential purposes
  - properties
  - kinds of users
  - integrated parts
  - architecture
  - etc.
- Explore the variables of each of those aspects.
- Identify and discuss product risks relating to the variables. Examples of risks that could be identified include:
  - accessibility risks
  - availability risks
  - charisma/likeability risks
  - compatibility risks
  - environment integration risks
  - functional risks
  - interface risks
  - localization risks
  - maintainability risks
  - observability risks
  - performance risks
  - portability risks
  - reliability risks
  - responsiveness risks
  - scalability risks
  - security risks
  - stability risks
  - testability risks
  - usability risks.

- Assess the risks and decide whether to invest further time and effort in mitigating or testing it. For more information on this topic, refer to the risk management practice guide.
- For significant risks, create a risk map. Risk maps are artefacts for service designers and developers. They also help to stem the test charters, which involves structuring exploratory testing by testing for specific risks in specific areas.

2.2.2 Testing in different environments

A risk-based approach is also helpful regarding test environments and deciding where to test.

Many risks can be tested for in a development environment. Development environments offer very quick feedback cycles because it is usually possible to quickly write code and use it to test for many kinds of product risks and then refactor the code if needed. However, certain risks cannot be tested for in a development environment; they might need a more stringently-integrated environment, such as a dedicated test environment.

Using a dedicated test environment is arguably slower because time is needed to create the environment and if any problems are discovered the feedback loop is longer in refactoring code, retesting in a development environment, then merging the fixes back in to the test environment again. Repeating tests that were completed in the development environment is unnecessary but testing for risks that could not be tested for in the development environment might be necessary.

Sometimes, having a pre-release environment (a staging environment) is sensible. Some risks may only be testable in a shared test environment, such as risks relating to data flows, platform risks, or some integration risks.

Finally, some risks are only testable in the real production environment.

Figure 2.1 represents the environments in which the majority of tests are performed.

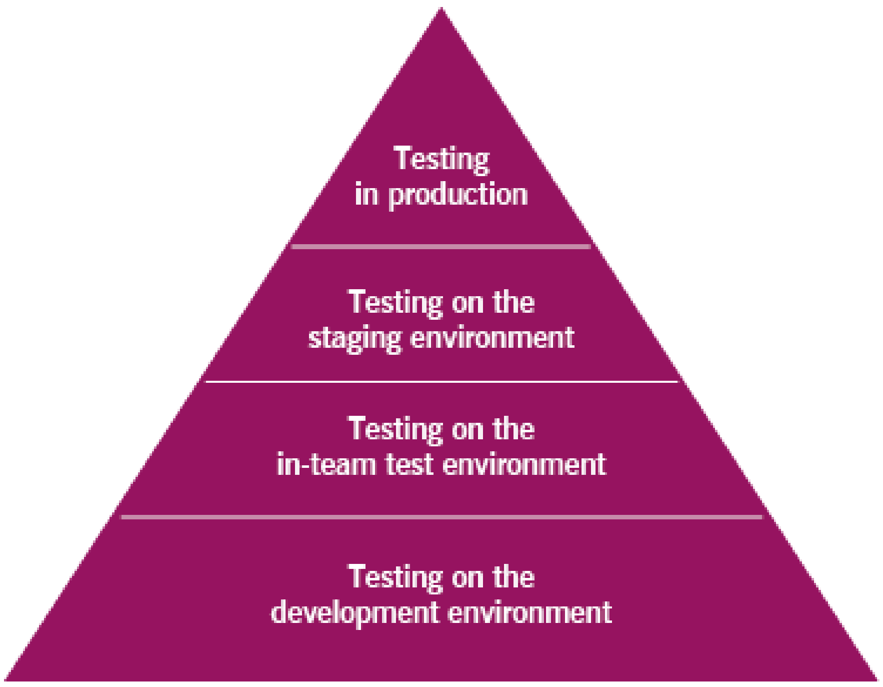
Testing in production
Testing on the staging environment
Testing on the in-team test environment
Testing on the development environment

Figure 2.1 Testing triangle

Most risks can be tested for early, in a development environment. Most of those remaining can be tested for in an in-team test environment. Most of those remaining can be tested for in a staging environment. Those remaining can be tested for in the production environment.

2.2.3 Assertive (scripted) and exploratory (investigative) testing

Testing provides information for decision-making in regard to a product or a service. Information is either known or unknown.

There are two states of known information:

- explicit information
- tacit information.

There are two states of unknown information:

- information that is known to exist, but which has not been accessed
- information that is not known to exist.

For further details on types of information and the related practical implications, refer to the knowledge management practice guide.

Based on the different states of information, there are two perspectives regarding software testing:

- Assertive (or scripted) testing aims to verify that a component, product, or service meets pre-defined criteria that are based on agreed requirements.
- Exploratory (or investigative) testing aims to uncover unknown information about a service component, product, service, or environment in order to identify risks that have not been addressed by pre-defined criteria.

The two approaches should be combined and balanced; adherence to one alone decreases the quality of information about products and services, which may lead to sub-optimal management decisions. Figure 2.2 illustrates how testing influences the information available.

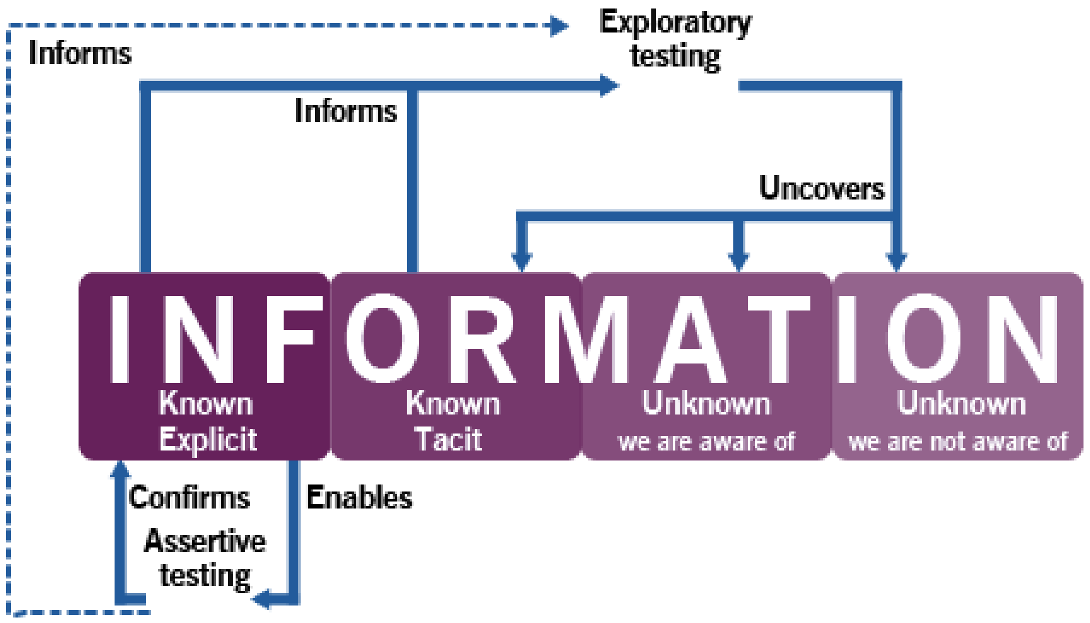

Figure 2.2 Testing helps to confirm and uncover information

2.2.3.1 Assertive testing approach

Assertive testing confirms whether explicit expectations on how a service should be designed, developed, and performed are being met.

This type of testing relies on explicitly expressed and documented expectations (usually in the form of the acceptance criteria). It also requires the tested artefact to be created.

Assertive testing can be performed manually or automated, depending on what is being tested and whether the organization has the required tools. Either way, assertive testing is based on documented test scripts that describe acceptance criteria, testing manipulations, and pass/fail criteria in a human or programming language. Automated testing is common for software and digital infrastructure, but it can be applied to other aspect of services, including controls, communications, system integrations, and interactions with users.

Assertive testing is limited by its nature: it can only be used to reduce known and documented risks in a limited spectrum of circumstances. It is also limited by the testing strategy for the product or service being tested; some known risks may be omitted to provide sufficient, not exhaustive, assurance.

2.2.3.2 Exploratory testing approach

Exploratory testing is based on investigating unknowns within a product, service, or environment with the intent of uncovering information that is relevant to the perceived quality and value of the services. It relies on lateral and critical thinking skills and is often based on the exploration of possible product vulnerabilities and associated threats.

Exploratory testing is commonly misunderstood to be ad hoc and unstructured. Actually, it is structured through small testing missions, known as test charters, which focus testing on targeted areas to investigate specific product risks.

This approach is essential in the context of agile development and the constantly growing complexity of information and organizational systems. It enables fast learning and feedback loops throughout the product and service lifecycle, which enables products' and services' continual improvement.

2.2.4 Continual validation and testing

The service validation and testing practice is not just about testing a releasable, operational product or service. These activities should be conducted throughout the entire service lifecycle, as was shown in Figure 2.2.

Validation and testing activities create important feedback loops, which inform every step of a digital product lifecycle, as Table 2.1 outlines.

Table 2.1 Validation and testing throughout a digital product lifecycle

Digital product lifecycle phase and associated artefacts	Validation	Assertive testing	Exploratory testing
Ideas	-	-	Exploration of the ideas and their relevance to customer's and organization's needs
Epics, user stories, features, enablers, and so on	Validation of epics, user stories, and features/enablers,	-	Exploration of epics, user stories, and features to identify

	development of acceptance criteria for UX/UI designs and architecture and code designs		inconsistencies and missed opportunities
UX/UI designs	Validation of the designs to confirm that they are based on the epics, user stories, and features and that they match the defined criteria Definition or update of architecture and code design acceptance criteria	Testing of the UX/UI wireframes to confirm that they meet architecture and design policies and guidelines, if applicable	Exploration of the UX/UI wireframes to identify inconsistencies and missed opportunities
Architecture and code design	Validation of the designs to confirm that they are based on the epics, user stories, and features and match the defined criteria Development of acceptance criteria for the code	Testing of the architecture and code design artefacts to confirm that they meet architecture and design policies and guidelines, if applicable	Exploration of the architecture and code design artefacts to identify inconsistencies and missed opportunities
Code units	Validation of the code to confirm that it was developed based on the agreed design, is complete, and adheres to agreed architecture standards Update of acceptance criteria for operational software	Automated, sometimes manual, unit testing to confirm that each unit performs as designed based on the agreed criteria	Peer review, pair programming, and other exploratory testing to identify errors and opportunities that have not been covered by the acceptance criteria
Operational software	Validation of the software to confirm that it is complete, based on the agreed criteria, and adheres to agreed architecture standards Development and update of acceptance criteria for deployment and release	Automated and sometimes manual unit testing to confirm that the software performs as designed, based on the agreed criteria	Review and exploration of the software to identify errors and opportunities that have not been covered by the acceptance criteria
Deployment and release pipeline	Validation of the deployment and release tools, processes, and methods to confirm that they meet agreed requirements and follow	Automated and sometimes manual testing of the pipeline tools and processes to	Exploration of the pipeline tools and processes to identify errors and opportunities that have not been

	the applicable policies and guidelines Update of acceptance criteria for deployment and release	confirm that they work as agreed	covered by the acceptance criteria
Software deployment	Validation of completeness and correctness of the deployment Update of the release acceptance criteria and regression testing criteria	Automated testing of the deployed artefacts to confirm that they meet the agreed acceptance criteria	Exploration of the deployed artefacts and environments to identify errors and opportunities that have not been covered by the acceptance criteria
Service release	Validation of the released service to confirm that it is complete and fits the agreed design specifications Update of the live service validation criteria	Automated and manual testing of the service operation, including user acceptance testing	Exploration of the released service to identify errors and opportunities that have not been covered by the acceptance criteria
Service in operation	Validation of service quality (utility, warranty, and experience levels) based on agreed criteria and service level management information.	Regression testing to confirm that previous test results are still valid	Chaos engineering to explore service vulnerabilities and other errors and opportunities that have not been covered by the acceptance criteria and formal service quality controls

2.3 SCOPE

The scope of the service validation and testing practice includes:

- translating the requirements for products or services into deployment and release management acceptance criteria
- establishing test approaches and defining test plans for new or changed products and services
- eliminating risk and uncertainty of new or changed products and services by testing
- discovering new information about new or changed products and services by testing
- continually reviewing test approaches and methods to improve the efficiency of the tests

There are a number of activities and areas of responsibility that are not included in the service validation and testing practice, although they are still closely related to service validation and testing. These are listed in Table 2.2, along with references to the practices in which they can be found. It is important to remember that ITIL practices are merely collections of tools to use in the context of value streams; they should be combined as necessary, depending on the situation.

Table 2.2 Activities related to the service validation and testing practice described in other practice guides

Activity	Practice guide
Establishing detailed requirements for the utility and warranty of the new or changed product or service Analysing new requirements for services outside of existing utility and warranty options	Business analysis
Maintaining financial control over testing Defining a testing budget	Financial management
Developing and managing software	Software development and management
Developing and managing infrastructure	Infrastructure and platform management
Operational communications with users and feedback gathering	Service desk
Deploying services and components	Deployment management
Releasing services	Release management
Ongoing management and implementation of improvements	Continual improvement

2.4 PRACTICE SUCCESS FACTORS

Definition: Practice success factor

A complex functional component of a practice that is required for the practice to fulfil its purpose.

A practice success factor (PSF) is more than a task or activity, as it includes components of all four dimensions of service management. The nature of the activities and resources of PSFs within a practice may differ, but together they ensure that the practice is effective.

The service validation and testing practice includes the following PSFs:

- defining and agreeing an approach to the validation and testing of the organization's products, services, and components in line with the organization's requirements for speed and quality of service changes
- ensuring that new and changed components, products, and services meet agreed criteria

2.4.1 Defining and agreeing approaches to the validation and testing of the organization's products, services, and components in line with the organization's requirements for speed and quality of service changes

Service validation should establish an approach to capture all of the utility and warranty requirements for any product, services, and components. This approach should involve different stakeholders and sources of information from the stakeholders, such as customer and user requirements and feedback, business requirements, internal and external compliance and regulatory requirements, risk and security, and other sources of requirements. This approach should also propose methods of translating requirements into acceptance criteria for the service.

A test strategy defines how testing should be implemented, considering the project’s objectives. Test planning should be based on the test strategy. The test strategy also defines the test management approach, including how testing will be organized and controlled.

The test strategy defines the test phases (or levels) and types that are in scope.

The testing phases include:

- **Unit** Undertaken by the developers to verify that what they have developed meets the requirements. A unit is typically a component of the overall system that is tested in isolation.
- **Integration** Undertaken when development is complete enough to start integrating different systems, concerned with the testing of the integration between systems.
- **System** Undertaken when it has been verified that the system’s components can be integrated, system testing considers the end-to-end functionality of the system.
- **Acceptance** User acceptance testing (UAT) is the formal test phase where end users verify and validate that what is to be delivered meets their requirements.

In each of these test phases, the test strategy must consider which test types are appropriate. Test types include:

- **Functional** Testing what the system being delivered will do.
- **Non-functional** Testing the aspects of the system that are not directly related to its functional requirements. Common non-functional aspects are:
  - **Performance** Behaviour under normal conditions.
  - **Load** Behaviour with increasing load.
  - **Stress** Behaviour when approaching the upper operational limits.
  - **Security** Authorization and authentication system controls.
  - **Usability** How well the users of the system can engage with the system.
- **Regression** New developments (progression) and bug fixes (debugging) can introduce unexpected system behaviour. Regression testing aims to verify that the system still functions as required following change.

Test plans define the detailed activities, estimates, and schedules for each test phase. Therefore, the test strategy defines the overall scope and approach, whereas the test plans detail each of the test phases. This is outlined in Table 2.3.

Table 2.3 Test strategy

	Test Strategy			
Types/levels	Unit	Integration	System	UAT
Functional	Unit test plan	Integration test plan	System test plan	UAT test plan
Non-functional				
Regression				

2.4.2 Ensuring that new and changed components, products, and services meet agreed criteria

No two projects are the same, and test strategies must be appropriate for the relevant project and organizational structure. Each test strategy should aim to:

- achieve the optimum test coverage in the time available by balancing effectiveness with efficiency
- be pragmatic and suit the needs of the programme, available resources, and available skills
- be aligned to the development methodology, technologies being employed, and the nature of the system being developed
- establish a high level of confidence in the delivery of software as early as possible
- confirm the accuracy of the software that is delivered (functional attributes)
- mitigate the level of business risk associated with implementing new software
- continue to improve and optimize the test process as the project unfolds
- identify test related risks, issues, and areas that are vulnerable and provide appropriate recommendations.

To achieve this, the test strategy needs to address:

- test organization
- test planning and control
- test analysis and design
- test preparation and implementation
- test progress and reporting
- incident management
- test closure and exit criteria.

The test strategy must consider the development methodology being employed. A waterfall development model often allows for the early static testing and validation of captured user requirements. A more iterative methodology may not deliver fully-formed user requirements before coding starts. The test strategy needs to be appropriate.

The test strategy must also consider the type of system/service involved. For example, testing a finance system at year-end requires a very different approach than testing an ecommerce website. It is important to consider the fundamentals of the testing environment: the processes, systems, resources, and management required to validate quality.

Testing is not limited to software artefacts; data migration, training, operational readiness, release management, and reporting are other areas that require specific testing attention.

2.4.2.1 Test organization

Those that test the system should be independent of those that develop the system. The mindset of a tester and a developer are different. Developers typically aim to prove that what they have developed meets requirements: testers aim to prove that requirements have been meet and that no other issues have been introduced.

The test organization should encourage this separation to improve the effectiveness of the testing. The roles and responsibilities of those involved in the testing should be clearly defined, including the test management and test analyst roles, as well as supporting roles involving incident management, configuration management, change control, deployment, and release.

2.4.2.2 Test planning and control

If the software development lifecycle follows a sprint-based methodology, testing should be involved in sprint planning. Each sprint should deliver artefacts that are testable, even if that requires the use of stubs and drivers, which should then be in the sprint scope.

Releases made available to testing consist of progression payloads (things that are new) and regression impacts (things that need testing to validate that they continue to function as required).

In terms of progression and regression, the threats to any release typically comprise of:

- New functionality being introduced to meet a requirement (progression and regression threat). This threat originates from the existing project.
- Bug fixes to new functionality (progression and regression threat). This threat originates from the existing project.
- Hotfixes to a production service (regression threat). This threat originates from the production service provider(s).
- Maintenance releases for a production service (regression threat). This threat originates from the production service provider(s).

2.4.2.3 Test analysis and design

Reporting on test progress only in terms of the percentage of overall coverage does not support informed risk assessment. To report on test progress in a meaningful way, testing should be aligned to the programme deliverables and requirements.

Each release to testing includes a payload. The payload can be divided into payload elements (PE). Each PE has a discrete test pack that is reported on.

For example, in a web-based order entry system, the programme schedule has defined a sprint to:

- PE 1: deliver the customer short code lookup facility (a progression deliverable with a regression impact)
- PE 2: allow payments to be made via credit card (a progression deliverable with a regression impact)
- PE 3: deliver the total order value as an automatically updated field on the order entry web page, thus replacing the need for the user to manually use the ‘calculate order total’ function (a progression deliverable with a regression impact)
- PE 4: up to ten development hours of bug fixes to be delivered in this sprint for open bugs from previous sprints (a progression and regression deliverable).

Testing has reviewed the programme schedule, obtained copies of the requirements and functional design documentation, and estimated that 45 test cases are required to cover all of the sprint’s progression deliverables. Additionally, when considering the regression risk of the sprint, testing identified a further 25 test cases that should be run as the sprint developments impacted the system’s core functionality.

This gives a total test pack size of 70 test cases if testing is planned for one functional test phase with two three-day cycles.

Reporting during the first test cycle might state that 80% of test cases have been run and passed by the end of the second day. However, this cannot be considered a good result without confirming which tests have been run and which are still to be run. Running and successfully passing all of the regression tests would indicate that the sprint introduced no regression issues.

Testing functionalities and recording successful results but not performing regression tests leads to high confidence in the new developments but does not indicate that the system has not been compromised. It is important to consider addressing the PEs and reporting on them upfront.

In the above example, knowing that there was one day left for testing in the schedule, ten regression tests outstanding, and five progression tests outstanding would inform the focus of the remaining test efforts.

To further subdivide regression tests, practitioners can consider the core functions of the system under test and define focus areas. Order entry could be one focus area, customer billing could be another. By categorizing regression testing across focus areas, a better assessment of outstanding test risk can be made.

2.4.2.4 Phases and cycles

Having defined the required test scope, practitioners can consider the test schedule by PEs and focus areas. Testing should begin as soon as the deployment and release activities to the test environment have been completed. It is important to consider the order in which the PEs and focus areas will be tested. The default should be to test the most complex or the newest developments, which are generally the highest risk, as soon as possible.

Having identified the testing scope estimation to test execution duration is required and given the defects impact test execution, defect rates needs to also be estimated and factored into a test execution schedule.

Planning the testing of a new system can be difficult; it is important to plan based on estimations of the outcomes and, as the system is iteratively tested, these estimates can be refined.

It may be useful to express the impact of defects in terms of test cases.

Example:

The next release of the latest sales order processing system has been analysed by the test team. Analysis of the PEs and focus areas has been completed. 172 test cases have been identified to cover the PEs, but the standard regression pack of 150 test cases and any additional 60 regression tests due to the nature of the PEs has been scoped, giving a total regression pack size of 210 test cases. In this example, regression tests are run manually.

At test planning, it is assumed that 20% of all PEs (34) and 10% of regression tests (21) will result in a defect.

Not all defects are equal; some are complex to triage and fix, and others are trivial. Defects are assumptively categorized as complex, standard, or trivial and assigned an amount of time that it should take to be resolved.

Table 2.4 outlines the information for an estimated defect total of 55.

Table 2.4 Example defect information

Categorization	Fix Rate	% assumed	Assumed defects	Weighting factor	Adjusted total
Complex	3 days	50	27 (50% of 55)	2	54 (27 * 2)
Standard	2 days	30	17 (30% of 55)	1.5	26 (17 * 1.5)
Trivial	1 day	20	11 (20% of 55)	1	11 (11 * 1)
Total					91

A weighting factor can be used to adjust for the complexity of some defects. The adjusted total can be treated as additional test cases to be executed. Adding these to the test case scope builds in an allowance for defect fixing.

Assumptions regarding how long it will take a tester to run a test are required for test execution planning.

Working on the basis of 'tests per tester per day' (TPTPD) can be useful. As with defect estimation, not all tests are equal: some taking longer to run than others.

Table 2.5 outlines the test case information based on the example test scope of 382 test cases (172 PEs and 210 focus areas).

Table 2.5 Results for the example 382 test cases

TPTPD	% assumption	Number of test cases	Test Duration for one tester
5	40%	153	31 days
3	35%	134	45 days
1	25%	96	96 days

Table 2.5 shows an allowance in the estimation of the duration of testing for full coverage. Including additional testers or scope reductions will reduce the test duration.

In the above examples, PEs and focus areas have been treated equally. Greater precision can be achieved by estimating these separately.

The test schedule should be organized by phase and into one or more cycles. Each cycle has a clear definition of the PEs and focus areas in scope for testing, with the highest risk generally tested first.

A 3-cycle approach for larger test phases is common. Table 2.6 outlines a 3-cycle approach.

Table 2.6 3-cycle test phase

Cycle 1	Cycle 2	Cycle 3 Final Test Cycle - FTC)
High risk/priority PE's	Lower priority PEs	Final fixes
High priority focus areas	Lower priority focus areas	Prioritized focus area clean run
	Fixes from Cycle 1	Prioritized PE backlog
	Backlog from Cycle 1	

2.4.2.5 Test preparation and execution

Planning for test execution can be a significant undertaking. Many factors need to be anticipated so that test execution can progress. Factors such as environment(s) provisioning, data creation, user account, and role configuration need to be planned.

A plan schedules the activities of and defines the roles and responsibilities for the test preparation phase. Daily stand-ups to monitor progress can be useful. Test preparation should be approached as a project in its own right, needing the usual project management techniques to ensure success.

Before test execution can begin, an agreed number of environment and system validation tests need to be successfully run. In addition, prepared test data should be closely guarded; it is often time consuming to generate test data. Data that has been created should only be used when the system under test can support the testing scope.

When test execution has started, it is important to ensure that momentum is maintained. Often, the initial stages of testing identify blockers; this should be anticipated. The appropriate resolver groups should be on standby with enhanced support to resolve early issues quickly. Often, resolver groups can remain occupied, such as by supporting ongoing development efforts for the next release of the system, while they are on standby.

Typically, the biggest threats to testing are incidents. To ensure focus is kept on the test execution, it should be the defect manager's responsibility to ensure that defects are resolved quickly and prioritized to support test execution progress.

As part of the defect management process, a clear definition of severity and priority is required. Severity, to measure the impact of the defect on the ability to release the system. Priority, to support the testing schedule. It is important to remember, though, that critical-severity defects are not necessarily highest priority.

If development are using a sprint approach, a number of hours per sprint could be allocated to test defect debugging (resolver group resources could be allocated to support testing). The testing defect manager should ensure that defects are debugged, fixed, and deployed for re-testing to support the test execution schedule.

The testing defect manager should monitor KPI's, such as the number of defects identified in the release, in order to identify areas where additional training and support may be required. KPI's focus on areas that have to potential to maximize improvement.

2.4.2.6 Evaluating exit criteria and reporting

Crucially, practitioners should know when to stop testing.

The exit criteria defined as part of the test analysis and design phase are used to identify when the system being tested is good enough. Test reports will reference the exit criteria and project an outcome, such as whether the test will finish on time. If the report indicates an issue, corrective action is required. Considering the test execution in terms of the PEs and focus areas is useful. In this way, reporting enables a more insightful view on the risk being carried.

If the same focus areas feature repeatedly in the regression testing for regular releases, normalizing their test execution schedules and comparing the progress of the test execution in this release to the last X releases clearly indicates the testing's trajectory.

At a minimum, daily and weekly test status reports detailing test execution coverage and pass/fail rates by PEs and focus areas (regression) should be required. These reports will provide metrics on the performance of the test capability, such as the actual observed TPTPD, defect rates by severity, debugging rates, and first-time fix rates and assess the trajectory of test execution.

2.4.2.7 Test closure

Once the final testing has been completed (the exit criteria has been met), test closure can be initiated. Depending on the nature of the system, this may be triggered at the end of testing. In other cases, this may follow a successful early life support (ELS) or hypercare phase following deployment and release. Often, key test resources are retained during a pre-defined ELS phase while the system stabilizes in the production environment.

This will involve:

- formally releasing resources from test activities and transitioning into business as usual (BAU) operations
- archiving and indexing test assets, including test strategies, plans, reports, and scripts
- transitioning to BAU will likely require reference to the test assets, especially when considering regression testing packs that the BAU support operation will need to maintain.

Lessons learned from testing should be captured, including both what did and did not work well. To support the Continual Improvement Practice, it is important to ensure that lessons learned are incorporated into the test strategy. Detailing which lessons are being addressed/repeated/not addressed. Those lessons not addressed and/or repeated should be transitioned by default to the next lessons-learned session.

2.5 KEY METRICS

The effectiveness and performance of the ITIL practices should be assessed within the context of the value streams to which each practice contributes. As with the performance of any tool, the practice's performance can only be assessed within the context of its application. However, tools can differ greatly in design and quality, and these differences define a tool's potential or capability to be effective when used according to its purpose. Further guidance on metrics, key performance indicators (KPIs), and other techniques that can help with this can be found in the measurement and reporting practice guide.

Key metrics for the service validation and testing practice are mapped to its PSFs. They can be used as KPIs in the context of value streams to assess the contribution of the practice to the effectiveness and efficiency of those value streams. Some examples of key metrics are given in Table 2.7.

Table 2.7 Examples of key metrics for the practice success factors

Practice success factors	Key metrics
Defining and agreeing approaches to the validation and testing of the organization's products, services, and components in line with the organization's requirements for speed and quality of service changes	Adherence to the service validation and testing approach(es) across the organization’s product portfolio Stakeholders’ satisfaction with chosen approach(es) to service validation and testing Stakeholders’ satisfaction with the organization’s ability to provide quality products and services Customers’ satisfaction with the products’ and services’ compliance to requirements
Ensuring that new and changed components, products, and services meet agreed criteria	Percentage of products and services meeting requirements for utility and warranty Stakeholders’ satisfaction with the chosen service validation and testing models and methods Stakeholders’ satisfaction with the organization’s ability to test products and services Losses from incidents and problems in services that are overlooked by testing
Aggregated metric for the practice	Service validation and testing productivity index[1]

The correct aggregation of metrics into complex indicators will make it easier to use the data for the ongoing management of value streams, and for the periodic assessment and continual improvement of the service validation and testing practice. There is no single best solution. Metrics will be based on the overall service strategy and priorities of an organization, as well as on the goals of the value streams to which the practice contributes.

[1] (N+C)/(O+C) – see the measurement and reporting practice guide for explanation and examples

3 Value streams and processes

3.1 VALUE STREAM CONTRIBUTION

Like any other ITIL management practice, the service validation and testing practice contributes to multiple value streams. It is important to remember that a value stream is never formed from a single practice. The service validation and testing practice combines with other practices to provide high-quality services to consumers. The main value chain activities to which the practice contributes are:

- design and transition
- obtain/build.

The contribution of the service validation and testing practice to the service value chain is shown in Figure 3.1.

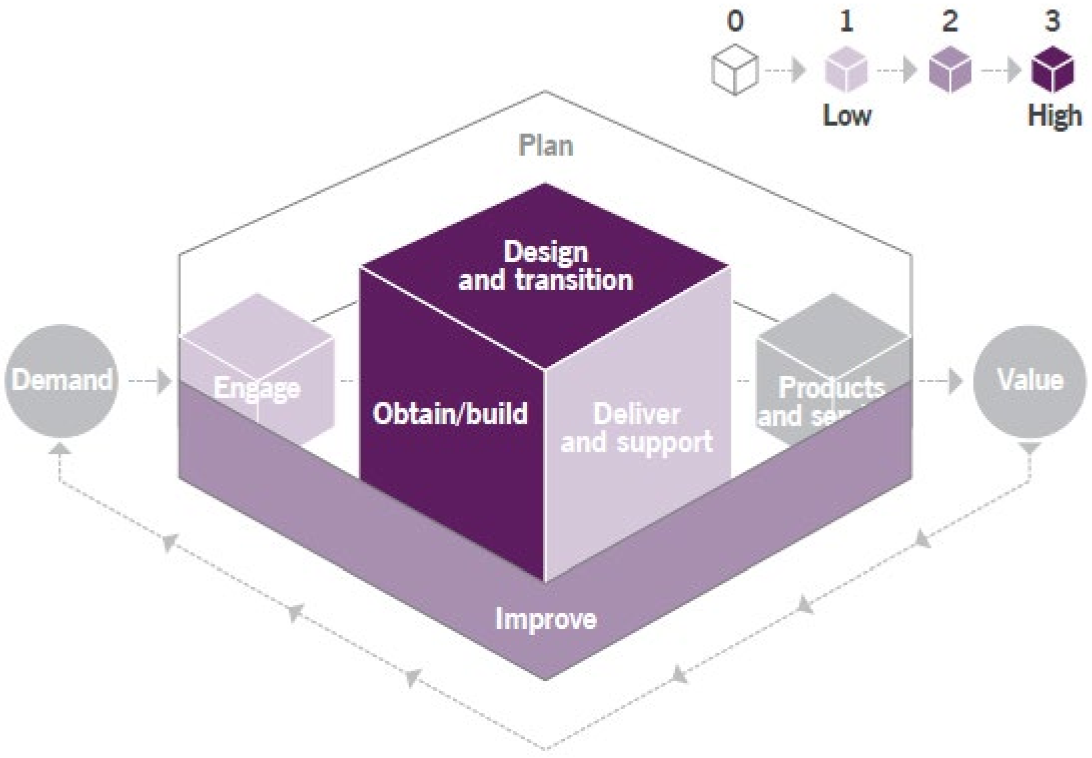

Figure 3.1 Heat map of the contribution of the service validation and testing practice to value chain activities

3.2 PROCESSES

Each practice may include one or more processes and activities that may be necessary to fulfil the purpose of that practice.

Definition: Process

A set of interrelated or interacting activities that transform inputs into outputs. A process takes one or more defined inputs and turns them into defined outputs. Processes define the sequence of actions and their dependencies.

Service validation and testing activities form three processes:

- testing approach and models management
- service validation

- performing a test.

3.2.1 Testing approach and models management

This process includes the activities listed in Table 3.1 and transforms the inputs into outputs.

Table 3.1 Inputs, activities, and outputs of the testing approach and models management process

Key inputs	Activities	Key outputs
Service models and design Updated release management approaches and models Release plans Existing test models, release models Updated release management approaches and models Release plans	Testing strategy definition and review Testing standards definition and review Test models definition and review	Test strategy and standards Test models, including test success criteria) Improvement initiatives Updated knowledge management articles

Figure 3.2 shows a workflow diagram of the process.

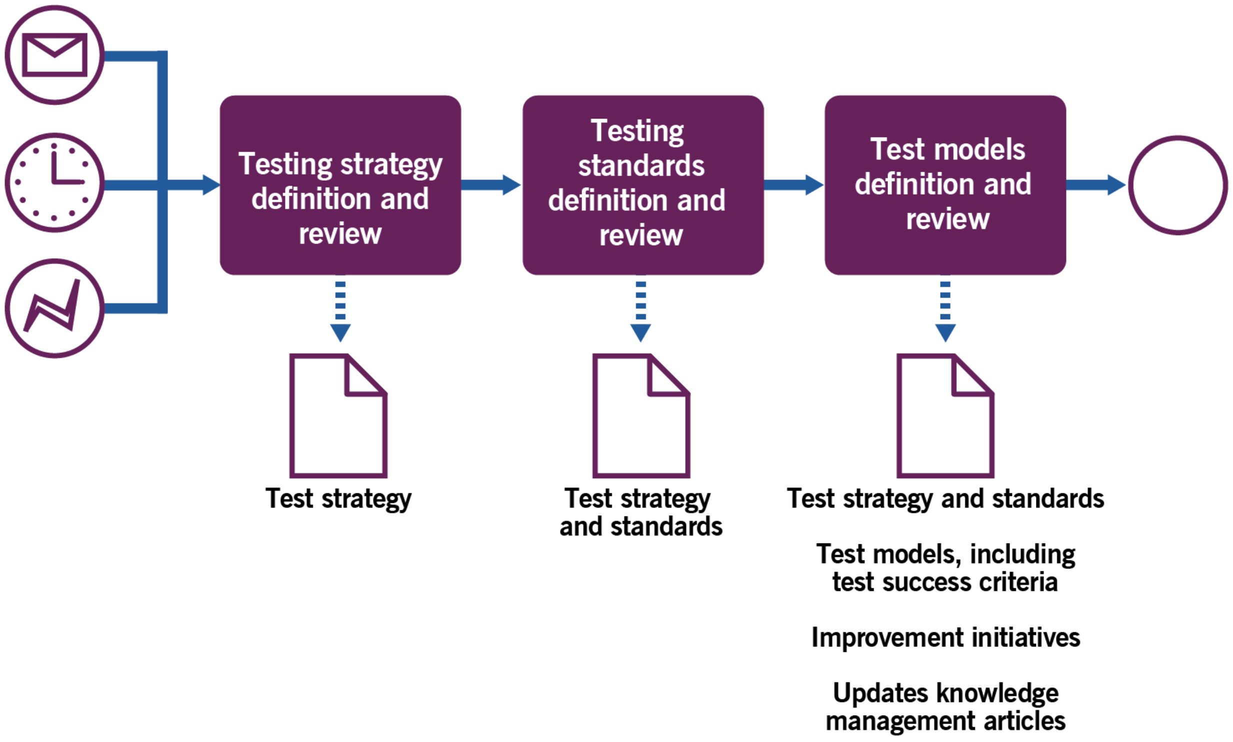
Testing strategy definition and review
Testing standards definition and review
Test models definition and review
Test strategy
Test strategy and standards
Test strategy and standards
Test models, including test success criteria
Improvement initiatives
Updates knowledge management articles

Figure 3.2 Workflow of the testing approach and models management process

Table 3.2 describes the activities in the testing approach and models management process.

Table 3.2 Sample description of activities in the testing approach and models management process

Activity	Description
Testing strategy definition and review	A service testing manager defines the testing strategy describing the approaches that a service provider organization adopts for testing and

	validation. The strategy establishes the baseline risk appetite of the organization and related testing efforts and resources. The testing strategy should be reviewed regularly to ensure the consistent achievement of quality goals.
Testing standards definition and review	A service testing manager defines standards for various types of tests that are applicable to different products and services along with the standards for recording the test outputs. Compliance to the standards should be monitored across all validation and testing activities.
Test models definition and review	A service testing manager establishes as-needed repeatable testing models to ensure consistent testing approaches for updated products and services. Otherwise, a test model can be produced specifically for a one-off large-scale service introduction in parallel with overall project planning activities.

3.2.2 Service validation

This process includes the activities listed in Table 3.3 and transforms the inputs into outputs.

Table 3.3 Inputs, activities, and outputs of the service validation process

Key inputs	Activities	Key outputs
Service design packages Utility and warranty requirements Test strategy and standards Test models Release plan	Documenting acceptance criteria	Acceptance criteria for a service Service testing scope and focus Service acceptance notice
	Verifying acceptance criteria	

Figure 3.3 shows a workflow diagram of the process.

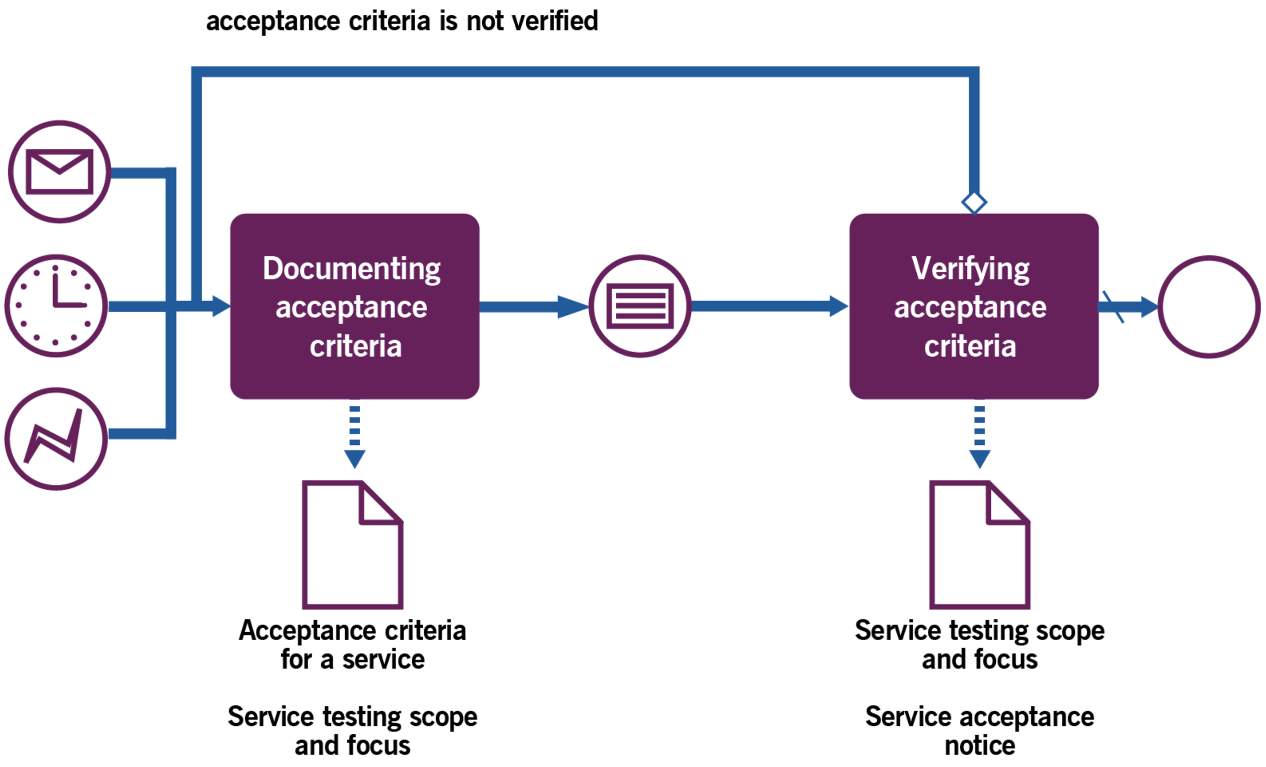
acceptance criteria is not verified
Documenting acceptance criteria
Verifying acceptance criteria
Acceptance criteria for a service
Service testing scope and focus
Service testing scope and focus
Service acceptance notice

Figure 3.3 Workflow of the service validation process

Table 3.4 Sample description of activities in the service validation process

Activity	Description
Documenting acceptance criteria	The service validation specialist establishes in conjunction with the service design practice and business analysis practice utility and warranty criteria that need to be tested for and met in order for a service and its components to pass tests. This activity occurs throughout the design phase of a service solution delivery.
Verifying acceptance criteria	A service validation specialist accepts test results and assures stakeholders that acceptance criteria have been met after a particular test. This activity occurs throughout the transition phase of a service solution delivery.

3.2.3 Performing a test

This process includes the activities listed in Table 3.5 and transforms the inputs into outputs.

Table 3.5 Inputs, activities, and outputs of the performing a test process

Key inputs	Activities	Key outputs
Release models Test models Acceptance criteria Testing strategy and standards	Test planning and preparation Test execution Test exit criteria evaluation and reporting Test closure	Configured testing environment Test and exit criteria report Lessons learnt

Figure 3.4 shows a workflow diagram of the process.

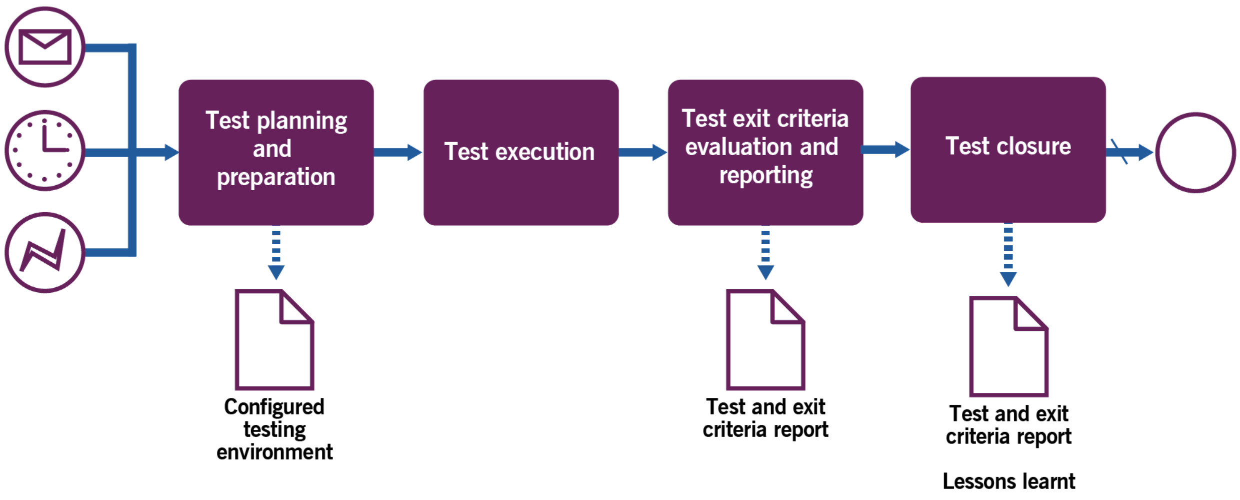
Test planning and preparation
Test execution
Test exit criteria evaluation and reporting
Test closure
Configured testing environment
Test and exit criteria report
Test and exit criteria report
Lessons learnt

Figure 3.4 Workflow of the performing a test process

Table 3.6 Sample description of activities in the performing a test process

Activity	Description
Test planning and preparation	A service testing manager reviews the acceptance criteria for the service or product being tested and plans environments, personnel, hardware, and other components that are required to perform a test, using the overall testing strategy, standards, and applicable models.
Test execution	A service testing specialist uses manual or automated tests and observes and records the outputs.
Test exit criteria evaluation and reporting	A service testing specialist reviews the results of a test and concludes whether success (or test exit) criteria were met.
Test closure	A service testing manager reviews test reports and formally authorizes completion of the test, if required by the test model.

4 Organizations and people

4.1 ROLES, COMPETENCIES, AND RESPONSIBILITIES

The practice guides do not describe the practice management roles such as practice owner, practice lead, or practice coach. They focus instead on the specialist roles that are specific to each practice. The structure and naming of each role may differ from organization to organization, so any roles defined in ITIL should not be treated as mandatory, or even recommended. Remember, roles are not job titles. One person can take on multiple roles and one role can be assigned to multiple people.

Roles are described in the context of processes and activities. Each role is characterized with a competency profile based on the model shown in Table 4.1.

Table 4.1 Competency codes and profiles

Competence code	Competency profile (activities and skills)
L	**Leader** Decision-making, delegating, overseeing other activities, providing incentives and motivation, and evaluating outcomes
A	**Administrator** Assigning and prioritizing tasks, record-keeping, ongoing reporting, and initiating basic improvements
C	**Coordinator/communicator** Coordinating multiple parties, maintaining communication between stakeholders, and running awareness campaigns
M	**Methods and techniques expert** Designing and implementing work techniques, documenting procedures, consulting on processes, work analysis, and continual improvement
T	**Technical expert** Providing technical (IT) expertise and conducting expertise-based assignments

Examples of roles involved in service validation and testing practice are listed in Table 4.2, together with the associated competency profiles and specific skills.

Table 4.2 Examples of roles with responsibility for service validation and testing activities

Activity	Responsible roles	Competence profile	Specific skills
Testing approach and models management process			
Testing strategy definition and review	Service testing manager	LMTA	Strong design-thinking Knowledge of testing methods and approaches
Testing standards definition and review	Service testing manager	LMCA	Knowledge of testing approaches and methods.

			Communication skills to enable standards compliance
Test models definition and review	Service testing manager	MTA	Knowledge of testing approaches and methods
Service validation process			
Documenting acceptance criteria	Service validation specialist	MTC	Knowledge of service validation approaches Understanding of business requirements
Verifying acceptance criteria	Service validation specialist	MTC	Knowledge of service validation approaches Understanding of business requirements
Performing a test process			
Test planning and preparation	Service testing manager	MACT	Strong resource planning skills Ability to plan with conflicting priorities
Test execution	Service testing specialist Service user (for UAT)	MT	Attention to detail Knowledge of testing methods and approaches
Test exit criteria evaluation and reporting	Service testing specialist	MT	Strong record-keeping skills Ability to clearly outline findings
Test closure	Service testing manager	MTC	Ability to align test results with the risk appetite as outlined in the testing strategy

4.2 ORGANIZATIONAL STRUCTURES AND TEAMS

4.2.1 Organizing for service validation and testing

Most service providers maintain a service validation and testing practice to ensure that their risk-based quality assurance approach is consistent. It is important to consider that testing (or often quality assurance) is the term most readily applicable to the software lifecycle; service validation is a broader area that includes products and service components beyond software, documentation, and digital infrastructure. Historically, this means that testing teams and validation teams are different: testing teams focus on application testing, validation teams are closer to service designers and architects. Both should work within the risk appetite that is outlined in the testing strategy.

4.2.2 Service validation specialist

This role can be fulfilled by service designers or architects to ensure that acceptance criteria, which are founded within the business or are technical requirements and constraints, are met during the tests, and that updated services and products also comply.

4.2.3 Service testing specialist

This is a core role within the practice, often called a ‘tester’ or a ‘QA engineer’. Their responsibilities include:

- conducting tests as defined in the test plans and designs
- recording and reporting on test results, including raising bug or incident records for unsuccessful tests
- administering test environments and associated resources.

5 Information and technology

5.1 INFORMATION EXCHANGE

The effectiveness of service validation and testing is based on the quality of the information used. This information includes, but is not limited to, information about:

- testing strategy
- testing standards
- test models
- test plans
- test records
- test results and reports.

This information may take various forms. The key inputs and outputs of the practice are listed in section 3.

The service validation and testing practice can significantly benefit from automation. Where this is possible and effective, it may involve the solutions outlined in Table 5.1.

Table 5.1 Automation solutions for service validation and testing activities

Process activity	Means of automation	Key functionality	Impact on the effectiveness of the practice
Testing approach and models management process			
Testing strategy definition and review	Resource planning tools Collaboration tools Analytical and reporting tools	Communicating the strategy and strategy updates	Medium
Testing standards definition and review	Resource planning tools Collaboration tools Knowledge management tools	Communicating the standards and standards' updates	Medium
Test models definition and review	Ticketing and workflow tools Knowledge management tools	Workflow design and tracking	High
Service validation process			
Documenting acceptance criteria	Collaboration tools Knowledge management tools	Record-keeping of acceptance criteria	Medium
Verifying acceptance criteria	Collaboration tools Knowledge management tools	Record-keeping of acceptance criteria	Medium
Performing a test process			

Test planning and preparation	Ticketing and workflow tools Knowledge management tools	Task planning	High
Test execution	Automated testing toolset and environments	Automation of testing and enabling	High
Test exit criteria evaluation and reporting	Ticketing and workflow tools Knowledge management tools	Processing workflow actions Record-keeping	High
Test closure	Ticketing and workflow tools	Processing workflow actions	High

6 Partners and suppliers

Very few services are delivered using only an organization's own resources. Most, if not all, depend on other services, often provided by third parties outside the organization (see section 2.4 of *ITIL Foundation: ITIL 4 Edition* for a model of a service relationship). Relationships and dependencies introduced by supporting services are described in the ITIL practices for service design, architecture management, and supplier management.

Commonly, service providers seek external quality assurance and testing capabilities to lessen the bias of testing naturally. Externally-managed service providers, teams, and individuals offer testing capabilities, services, and products in the software domain as well as in specific non-functional testing areas. However, a holistic validation of services that are deployed to specific service customers often warrants a service provider organization to foster an internal service testing and validation practice to ensure the holistic coverage of acceptance criteria during a service introduction journey.

Where an external commercial service provider manages the service delivery value stream, a customer organization may seek additional expert service deployment capabilities to ensure that their requirements are being met. The assurance services of external and, most importantly, independent service validation and testing providers are also available.

Where organizations aim to ensure fast and effective service validation and testing, they usually try to agree close cooperation with their partners and suppliers, removing formal bureaucratic barriers in communication, collaboration, and decision making (see the supplier management practice guide for more information).

7 Important reminder

Most of the content of the practice guides should be taken as a suggestion of areas that an organization might consider when establishing and nurturing their own practices. The practice guides are catalogues of topics that organizations might think about, not a list of answers. When using the content of the practice guides, organizations should always follow the ITIL guiding principles:

- focus on value
- start where you are
- progress iteratively with feedback
- collaborate and promote visibility
- think and work holistically
- keep it simple and practical
- optimize and automate.

More information on the guiding principles and their application can be found in section 4.3 of *ITIL Foundation: ITIL 4 Edition*.

8 Acknowledgments

AXELOS Ltd is grateful to everyone who has contributed to the development of this guidance. These practice guides incorporate an unprecedented level of enthusiasm and feedback from across the ITIL community. In particular, AXELOS would like to thank the following:

8.1 AUTHOR

Peter Bodman and Dan Ashby,

8.2 REVIEWERS

Roman Jouravlev and Dinara Adyrbai