Software Testing and Quality Assurance Guidelines

These software testing and quality assurance guidelines aim at guiding and addressing concerns on software quality processes during the acquisition, development, and maintenance of systems by developers, managers, and quality assurance teams.

    - Risk Based Approach to Testing
    - Software Testing Guidelines Objectives
    - Roles and Responsibilities
    - Fundamental Test Planning Process and Strategies
    - Approaches to Manual Testing
    - Approaches to Automated Testing
    - Defect Management

- Test Data Management
    - Purpose of the Test Data document
    - Ways of Collecting Test Data
    - Practical guideline on how to use test data

- Test Environment Management
- Performance and Security testing
    - Performance testing
    - Security Testing

- Test status and reporting
    - Statuses
    - Evaluation of test processes and test improvement (Continuous Improvement)
    - Project Lesson Learned workshops

- Compliance and Audit
- Test control and review history
    - Test control
    - Review History

- Requirement and Traceability Matrix

Introduction

These software testing and quality assurance guidelines aim at guiding and addressing concerns on software quality processes during the acquisition, development, and maintenance of systems by developers, managers, and quality assurance teams.

Purpose of the document

This document is divided into two sections and describes the two key elements to the approach and vision of testing within the GoR Institutions:

- Software Testing Policy: An overview of the values supporting the testing approach
- Test Guidelines: These are best practices on how testing should be done, what needs to be taken into account, and the different test stages that may be used to ensure that systems are tested properly in order to create dependable and high-quality systems.

These guidelines will be revised from time to time based on the latest technology trends, feedback from Government Institutions and the level of maturity of the testing services functions.

Overview

RISA is driving the government’s digitalization and innovation plan by supporting sectors and institutions in their automation journey.

Automation of services and processes involves acquiring or developing software. In order to ensure that the software meets the requirements and will serve the intended purpose, software testing guidelines were developed to provide guidance and best practices on the testing process.

Scope

This document provides policies and guidelines for the planning and execution of software testing activities for Government of Rwanda Institutions. It is crucial that the consumers of this document (i.e., software project teams in Government of Rwanda Institutions) not view these strategies as mandatory standards, but rather as a reference model and adopt the recommendations in accordance with the specifics of each project.

The guidelines are meant to be relevant to both internally generated and externally contracted software development projects.

Software Testing Policy

Purpose of this Software Testing policy

The testing policy outlines the high-level strategy and principles for software testing. It outlines the test goals and objectives, benefits and strategies for test process improvement. All subsequent test strategies and low-level test plans will be established on the basis of this policy.

Mission Statement

To ensure that any changes to software do not jeopardize the quality characteristics of mission-critical systems used by GoR Institutions.

To verify, validate, and evaluate the quality of software or systems being delivered and to ensure that testing is performed effectively and dynamically to ensure that the product satisfies the end-user requirements resulting in the institution’s satisfaction.

Software Testing Policy Objectives and Principles

The following are the main objectives of the software testing policy:

1. To ensure that testing is planned early in the project so that testing activities may be started as soon as possible in the project life cycle, and resources planned accordingly.
2. To identify and include important stakeholders in the testing process, including developers, technical advisors, and software testers, in order to understand testing priorities and risks from both a business and technological perspective.
3. To ensure that a risk based approach is used to prioritize testing activities with the added benefit of making optimal use of resources by focusing on the most critical areas of testing.
4. To ensure defect discovery whether it be requirements defects, code defect or other types of defects, occurs as soon as feasible to avoid spiraling expenses associated with finding and correcting issues later on in the project/software's lifecycle.
5. To ensure that important business needs are covered by testing through comprehensive traceability of testing coverage back to original business requirements.
6. To ensure there is focus on testing the 'unwritten requirements' of a system to identify any defects that users may find. Unwritten requirements cover areas such as user behavior which involve using the system in the context of the end users rather than how it has been designed.
7. To ensure that testing also covers compliance with applicable policies and regulations

Benefits of the Software Testing Policy

This testing policy aims to give the GoR institutions the observable advantages listed below:

1. Ensures that testing is a standardized and repeatable activity rather than a customized activity for each project or software modification, i.e., that we are not reinventing the wheel but also not restricted by a single restrictive technique. The mindset that this policy and accompanying approach will create is a "freedom to test" mentality.
2. Supports successful collaboration across all GoR institutions through the adoption of standardized concepts, methods, and deliverables.
3. Provides a framework for defining how test planning, design, execution, and closure are conducted.
4. Ensures adequate resources for software testing are provided to ensure that testing is effective.
5. Ensure that regression testing is carried out which ensures that existing processes and functions are validated when new fixes are introduced in software during the testing process.
6. Ensures completeness of testing by relating testing to both functional and non-functional requirements such as security and performance testing.
7. Ensures that all testing operations are documented to an agreed-upon, suitable degree in line with the testing policy.
8. Ensures that the available resources are used as efficiently as possible to carry out testing activities.

Testing Guidelines

Purpose of Software Testing Guidelines

The major purpose of these software guidelines is to provide a set of recommendations and best practices for ensuring that systems are properly tested to achieve high quality software solutions.

These guidelines explain the basic framework of the testing process to project managers, team leaders, testers, and developers. This covers the testing goals, techniques for testing both new and current features, testing roles and responsibilities and defines the kinds of tests that should be run. Unless specifically stated otherwise, the testing method must be followed for all projects where testing is necessary.

The RISA Software Solution Division will need to authorize the deviation in this case , which should be noted in the lower-level test plan.

Definition of Testing

Below is a description of how testing is defined generally.

General Definition

Testing is the process of executing program(s) with the intent of finding errors, rather than (a misconception) showing the correct functioning of the program(s). The difference actually lies in the different psychological effect caused by the different objectives: If the goal is to demonstrate that a program has no errors, then we will tend to select tests that have a low probability of causing the program to fail. On the other hand, if the goal is to demonstrate that a program has errors, the test data will have a higher probability of finding errors.

Objectives of Testing

The objectives of testing are:

1. to verify and enhance the product's end-user experience and make sure it is fit for its intended usage.
2. to test as early in the software development lifecycle as feasible (for example, in the requirements definition stage) utilizing various methodologies (for example, testing unwritten requirements such as user behavior of the system). The goal is to prevent defects from occurring in the first place, thus finding any defects is not sufficient.
3. confirming that a product meets its primary needs. confirming and approving this claim by repeatedly asking, "Is this what the end user requires?"

Risk Based Approach to Testing

All facets of the testing lifecycle should be approached using a risk-based approach. Testing should always be done with a view to identifying and reducing any risks the Project/Change may present, depending on the project objectives. The necessary testing will take into account any such risks that are discovered. In layman's terms, the test effort should start with the area where one component of a system is thought to have a potentially high-risk consequence if it fails.

The project's appointed test lead should decide on the appropriate level of quality risk reduction. The test effort and test sequencing will depend on the risk level. This approach allows test results to be presented in terms of both mitigated and unmitigated risks. A project's or change's testing scope should consider a number of criteria. Depending on the scale and scope of a particular project, a combination of those outlined below may be used:

1. Analyze the potential product risks and concentrate testing on the high-risk areas (also known as risk-based testing).
2. The most frequently used functions can be reviewed, and testing could be concentrated on evaluating these areas, especially if the system under test is already in place and a change is thereby being implemented.
3. Determine the effort of testing required based on the project criticality. Depending on how crucial the project is, different testing levels will be used.

Software Testing Guidelines Objectives

The key principles that should guide the development of strategies for all testing activities are described in the following testing objectives:

1. To ensure creation of a testing plan early on in the project so that resources may be allocated appropriately, and testing activities can begin as soon as possible.
2. To understand testing priorities and risks from all perspectives of the business and technology. This can be achieved by identifying and including key stakeholders in the testing process.
3. To ensure there is communication with the appropriate parties on a regular basis, providing updates and inviting them to take part in important checkpoints where clarification is needed on things like needs.
4. To implement a risk-based testing strategy to help prioritize, target, and concentrate testing with the added benefit of making efficient use of resources.
5. To avoid spiraling costs associated with finding and repairing issues later on in the project/software lifecycle, put more emphasis on making sure the defect discovery component of testing occurs as soon as is practically possible.
6. To ensure that every test coverage can be fully traced back to the original business requirements in order to guarantee that testing has covered all crucial business requirements.
7. In compliance with the Testing Policy, record all testing operations at an agreed-upon, acceptable level.
8. To utilize resources as effectively as possible when conducting testing activities.

Roles and Responsibilities

The table below describes the three aspects of roles and responsibilities of personnel that are typically required to be involved with testing for software. The following defines the three aspects:

Standard Testing Roles

This is an industry standard testing role that a member of staff will directly or indirectly inherit as part of the project/change. Typically, roles and responsibilities should be agreed at the very start of a project and documented. It is recommended a 'Test Initiation Meeting' is held by the Project Manager / Team Leader to define and assign roles and responsibilities.

RISA or Government Institution Mapped Roles

This describes a role which currently exists in RISA or a Government institution which will typically take on the relevant Testing Role (s) for that project. Whilst not exhaustive this is used to illustrate the type of roles in GoR institution as well as RISA that may map to a Standard Testing Role for a project. It may well be that **one person** in a Government institution may undertake **multiple Standard Testing Roles** and therefore assigning 'real life' roles to testing roles is **not a 1-to-1 mutually exclusive scenario.**

Responsibilities

This guideline provides an overview of the standard roles and responsibilities that are typically required during software testing. It is recognized that it may not be practical to have dedicated individuals performing each of the standard roles. In this case Government Institutions can assign these responsibilities to existing project members on a project-by-project basis. The table below summarises the recommended roles with an example provided of how RISA assigns the standard roles to existing team members.

Key testing tasks and activities that the testing role incumbent would perform.

Standard testing roles	RISA team members who can be assigned the standard role	Responsibilities

Test Manager	RISA Test Manager RISA Test Lead Test Team Member	● To provide guidance to the test leads and teams on the framework, approach and processes the test teams should adopt for the project. ● Monitoring of ongoing projects and milestones, providing inputs and approvals at quality gates. ● Assisting in creation and production of test plans and providing ongoing testing related training.
Test Lead	Unit & System Testing Team Leader User Acceptance Testing: Senior Business Partner Representatives Test Team Member	● Leads a team handling a specific component of testing such as a module of the system. For small projects Test Manager and Test Lead can be performed by the same individual. ● Day to Day management of test activities required for that test phase / project. ● Recruitment of test analysts to support the design and execution stages of that test phase. ● Responsible for ensuring Test Process deliverables (e.g. test scripts) are produced within the agreed project timelines. ● Regular reporting of progress during all stages of the Test Process including Planning, Design, Execution and Closure.

Test Analyst	Developer UX team member Business Analyst GoR Institution/ RISA Staff	● Designing Test Scenarios and/or Test Cases as part of the Test Design Phase. ● Identifying Test Data required to support the tests during execution and working with support teams to produce the data. ● Execution of the tests and providing results to the test lead and raising any defects found during execution.
Defect Manager	Project Manager Team Lead Test Team Member	● Responsible for analysing, categorizing and tracking defects to ensure they are fixed in a prioritized manner. ● Tracks and reports on key metrics on open and closed defects. ● Arranges and leads triage meetings between developers and testers when required to assist in resolving high priority defects. ● Obtains approval for deferral of any open defects found to exist prior to 'Go-Live'

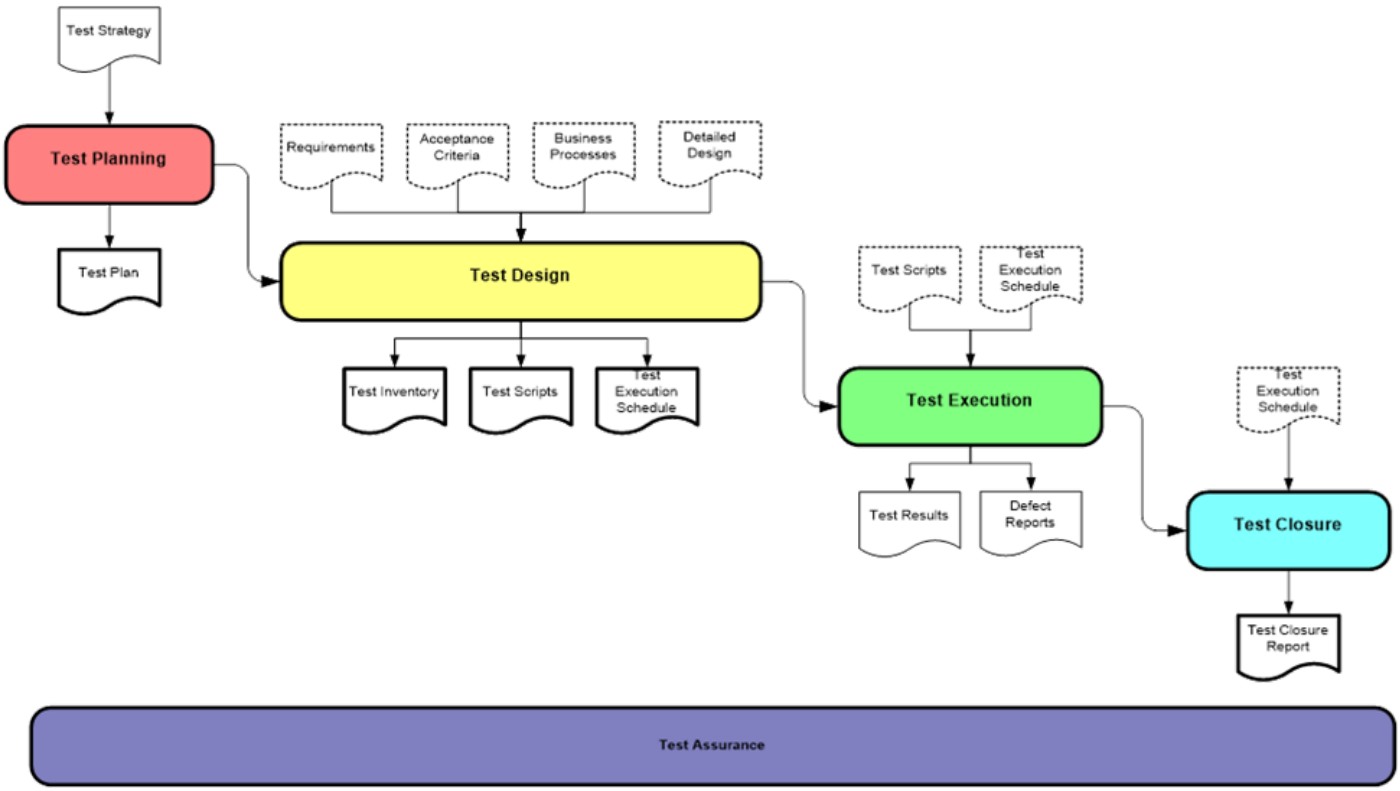
Fundamental Test Planning Process and Strategies

The four stages of the Fundamental Testing Process must be completed in the correct order. These are Test planning, Test Design, Test Execution and Test Closure. These stages should be carried out in this logical order for any Test Level i.e Unit, System, System Integration Testing and User Acceptance Testing (UAT)) to maximize the benefits to the testing process and overall quality assurance.

Whilst at first glance using the process may appear excessive for some projects, the stages can be as formal or informal as required. This underpins the Context-Driven-Testing philosophy which states, “The value of any practice depends on its context”. As long as the basic principles of the process are being followed then the testing process will produce the required result. The test process may vary in terms of duration and activities involved depending on the development methodology being used by the development teams. For example, Application Development are very much Agile in their development methodology while other areas such as research Systems are very much traditionally waterfall / V Model in their development methodology.

Below is an overview of each stage of the Fundamental Testing Process and also the test deliverables that should be produced as the outputs of the stage.

1. Test Planning
2. Test Design
3. Test Execution
4. Test Closure



To aid and assist in each of the test stages, the following sections provide some guiding principles on how to ensure each stage is performed effectively and to obtain the optimum benefits (both to testing and the wider project delivery).

Test Planning

The objective of test planning is to prescribe the scope, approach, resources, and schedule of testing activities for a level of testing. It also includes identifying the items being tested, the features to be tested, the testing tasks to be performed, and the personnel responsible for each task.

Make sure the resources needed to support the test design and execution phases are decided upon and assigned early enough to allow for sufficient test preparation.
To do this, a Test Initiation Meeting should be held at the very beginning of the Project.
For each project, a test plan should be prepared and evaluated with end users, user groups, and all other relevant stakeholders whenever practicable (and practical). At least the following steps should be respected:

1) First, analyze product structure and architecture.
2) Design the test strategy.
3) Define all the test objectives.
4) Define the testing area.
5) Define all the usable resources.
6) Schedule all activities in an appropriate manner.
7) Determine all the Test Deliverables.
8) Collapse your test plan, avoid overlapping and redundancy.

9) Update plan when needed and do not use an outdated and unused document

Testing risks and Mitigations

Any risks that will affect the testing process during a Project / Change should be listed along with the mitigation. By documenting a risk, its occurrence can be anticipated well ahead of time and therefore proactive action may be taken to prevent it from occurring, or to mitigate its damage. An example of a risk to testing could be that a particular testing tool required for that project may not be available. Any such risks to testing should be documented in the Test Plan for that test level / project.

Test schedules

The length of time it will take to complete the testing phase should be estimated. The test schedule for the particular test level should be formed from the estimations and included in the test plan. Scheduling for testing should factor in the following:

- First of all, testers are required to run each test case at least once (usually covering new features and regression testing).
- Additionally, if a flaw is discovered, the developers will need to resolve the issue. The failing test case should then be retested until it is working properly.
- Last but not least, the tester must perform further regression testing toward the end of the testing cycle to ensure that the software's other components have not changed.

Test priorities

As part of a risk -based approach to testing, when designing test cases it is important to assign priorities for each test or group of tests. When testing is required on projects, certain test cases will be treated as the most critical tests and if they fail, the product cannot be released. Some other test cases may be deemed less critical if they were to fail. As part of the Fundamental Test Process, the test design activity will involve creation of test cases that will form 3 test methods:

1. Verify the new functionality / changes being introduced (verification testing). This involves testing of written requirements
2. Attempt to find any unexpected behaviors in functionality / changes being introduced (defect discovery testing). Known as testing the unwritten requirements
3. Verify that existing functionality continues to behave and perform as normal following introduction of changes (regression testing)

Each test should have a priority weighting attached to it:

- High – Any tests deemed to be in this category mean they must be run initially as the functionality being tested is critical and if failed would have a severe impact
- Medium - Any tests deemed to be in this category mean they must be run only after the High category tests. The functionality being tested is important and if failed would have a high but not severe impact

- Low - Any tests deemed to be in this category mean they must be run only after the medium category tests. These are effectively ‘nice to have tests. The functionality being tested is not critical important and if failed would only impact a small volume of users whilst at the same

It is worth mentioning that the weightings above must, where possible, be agreed with key stakeholders, especially the stakeholder who has requested the Project / Change.
Whilst this may seem like overkill, this is introducing a risk-based view to test coverage and if the time allocated for testing is reduced the high priorities areas would be able to be covered first.

Test case design

Different projects may require different levels of test documentation. Ensure any test cases that are created are done so only if they are going to add value.

Not every project will require the creation of large volumes of test cases, but practically every project will need some kind of documented testing.

Before testing starts, make sure tests have been designed, evaluated, and authorized. Test cases ought to include:

1) The reference to the requirement being tested. This is to ensure traceability of requirements.
2) Description of the function or feature being tested.
3) Name of person who designed the test & date.
4) Test data & instructions to run the test
5) Expected results.
6) Level of priority as detailed above

Testers should approach all testing with a "test to break" mentality while planning and carrying out tests, with the goal of identifying any faults.

- It is crucial to keep in mind that tests should be designed to test both the specified and unwritten requirements (including usual user behavior and arbitrary scenarios that could push the system to its breaking point).
- Test Scenarios should be the first part of the test design process. Scenarios should be captured in a “Test Scenario Inventory”.
- Scenarios should be kept very short and used as a high-level view to describe the function or process to be tested.
- Scenarios should be written so that they can be easily reviewed by anyone wishing to understand what is being tested.

Test execution

Before starting any Test Execution phases where testers may be unfamiliar with the testing process, it is worthwhile to have a preliminary informal meeting to go through the key expectations such as:

1. How the testers will record the results. This includes agreeing on any automated tools for tracking testing and recording results.
2. Walkthrough of some example test cases to make them feel more comfortable. This will avoid a lot of downtime during the testing window with testers asking questions.
3. How the testers should record defects. Walk through the process of how issues will be reported and what information is required.
4. How long the testers have to complete the testing. This will help testers focus and plan execution of their test cases within the available time.
5. Who will be required to support the testing activities. For example developers may be required to support testing in the event issues are identified that block testing progress. Similarly this also ensures the required resources from the relevant business units or subject matter experts are available.
6. Location for the testing activities. Attempt to bring the testers together in one room or location where possible.
7. Test progress reporting. Detail what should be reported on, frequency and audience..

Test closure

Make sure to run all the tests that can be executed.

Make sure you have justifications for any tests that cannot be completed in writing.
Take the time to analyze the results independently and make sure you are comfortable with the message you will be conveying before any Go/No Go quality gates at the final stage of testing.

After testing is completed, look at any open defects.

1. If any defects are open, then reasons for defects being open will need to be given. The defects should be analysed and impact on the software and operations documented. Based on this a decision should be made on which defects must be closed before the project or change can go to Production
2. Stakeholders will understand these points as long as you have documented reasons.

Test levels

Utilizing test levels, or test phases as they are more frequently known, encourages quality risk mitigation as early as feasible and to the fullest extent possible. The table that follows lists the different test levels that, unless otherwise specified in the applicable Test Plan, should be carried out for all Projects and Changes.

Note: There should be a specific representative assigned to each Test Level who is in charge of carrying out or supervising the activity.
The notable exception to this rule is the System Integration Test Level, which is only necessary when a project or change introduces changes that have an impact on multiple systems or a system's interface with another system.

There are 4 levels of Testing, each of which carries a specific functional purpose, to be carried out in chronological order.

Level	Owner	Objective	Key areas of testing
Unit	Development Team	● Detect defective code in units ● Reduce risk of unit failure in Live Service	● Functionality ● Resource utilization
System	Development Team	● Detect defects in end-to-end scenarios concentrating on systematically validating whether each function performs as expected or not ● Assist in mitigating risk of unmet business requirements	● Functionality ● Performance ● Reliability ● Usability ● Resource utilization, ● Maintainabil ● Installability, portability interoperabilit
System Integration	Development	● Detect defects in unit interfaces ● Reduce risk of dataflow and workflow failures in Production	● Functionality ● Data quality ● Unit interoperabilit ● Compatibilit ● Performance
Acceptance (User, Business, Operational)	Senior Business Partner (who requested the deliverable of the Project / Change	● Demonstrate the product works as expected in a real-life environment using real life operational scenarios ● Detect defects in user workflows ● Ensure key business acceptance criteria are met	● Functionality context of nor usage ● Operational Processes ● Pre-agreed acceptance cri

Regression testing

The purpose of regression testing is to verify that one fix or change does not inadvertently impact other areas in that program or in any other interface.

When considering any form of testing it is strongly recommended that regression testing is always considered when planning the scope of the test level. It is the intention of these guidelines to put regression testing at the heart of the testing design and execution process.

The following statements describe the approach to regression testing that should be undertaken:

1. Regression testing of a brand-new system is not possible initially. However once a new system has a defect and associated fix applied, a set of focused regression tests should be performed not only on the failed area but also areas surrounding that function.
2. If the Project / Change is amending or modifying an existing system, then a high degree of regression testing should be performed.
3. Regression testing should not be the last element of any testing that is required. If the requirement for regression is high, then as well as initially focusing on the new functionality during test execution, a focused set of regression tests should be planned.
4. Where possible, a set of regression tests should be designed and maintained for each mission critical system. The regression tests should each be prioritized using the following weightings:

    - High – Any regression tests assigned to this category are required to be executed as early on in the testing window as possible. This classification is used where functionality being tested is critical and if it does not perform as required would have a severe impact.
    - Medium - Any regression tests assigned to this category are required to be executed only after the High category tests. The functionality being tested is important and if failed would have a high but not severe impact
    - Low - Any regression tests assigned to this category mean they must be run only after the Medium category tests. These are effectively ‘nice to have’ tests. The functionality being tested is not critical important and if failed would only impact a small volume of users

Approaches to Manual Testing

Manual testing is an essential part of the testing approach, given the volume and diversity of the applications that will be developed, supported and utilized by GoR.

There are three test techniques to undertake manual testing which have been detailed below. Each technique can be utilized as appropriate. However, the points below give a standardized suggestion of which types of Projects / Changes these can apply to. Referring again to the philosophy of Context- Driven Testing, this is not meant to be a prescriptive approach to testing. The objective is to provide test representatives ideas which they can apply to their own project using one technique or a combination.

1. **White-Box Testing** - also known as Code Testing, focuses on the independent logical internals of the software to assure that all code statements and logical paths have been tested.
2. **Black-Box Testing** - also known as Specification Testing, focuses on the functional externals to assure that defined input will produce actual results that agree with required results documented in the specifications.

    - Typically used when testing Web Based Applications and /or systems where there is a set of clearly defined and document requirements and associated design specification
    - This is the traditional method of designing test scenarios and test cases in advance of performing tests, ensuring an expected behaviour is defined in the test scenario and / or the test case.
    - During the execution of each test, the defined test steps (what the tester is required to follow to perform the test) would be followed systematically without diverging away from what the test is asking the tester to perform.
    - If the expected result matches the actual result (what the tester has observed) then the test would be marked as Passed.
    - If there is a difference between the expected result in the test and the actual result, then the test would be marked as Failed (regardless of suspected root cause).
    - Once a test is marked as Failed a defect would then need to be recorded and passed to the development team for triage and fixing.

3. **Exploratory Testing** - This is a relatively new technique that is utilized when the system under test has too many features to be systematically tested given the project's potential time and resource limits, or when the system can be accessed in multiple ways on multiple platforms by multiple devices.

As part of these guidelines, it is recommended that the exploratory technique is adopted particularly for any Projects involving Mobile application development.

Exploratory testing removes the dependency to create a list of cases in advance of performing tests, rather the testers explore and navigate around the system or software application.

It should be noted however that where possible script-based testing should be used in conjunction with exploratory testing.

Approaches to Automated Testing

The focus of these guidelines and the initial remit of GoR institution Testing Services is aimed at introducing and embedding the key principles of manual testing to GoR Institutions and associated business areas. Once the principles and associated processes of manual testing have been implemented and suitably matured the aspirational goal is then to look at how automated testing can complement and replace existing manual testing processes.

However, whilst automation of testing activities will not formally be part of the day-to-day processes, GoR Testing Services will always be reviewing ways to improve the testing framework. Further regular reviews of this policy are planned and the approach to automation may be updated to reflect the approach at that time.

Defect Management

The way in which defects are handled is arguably equally as important as running the tests themselves.

The basic process of defect management is described below and should be followed whenever there is a test phase being undertaken.

Step 1: Defect observed

- Tester observes a difference between the expected result defined in the test and the actual result demonstrated by the system.
- Tester needs to ensure the requirements have not changed. If they have then this means the test is factually incorrect and requires to be updated. No defect required if this is the case.
- Tester also needs to check if the defect has been previously reported by another tester or if the development team is aware of this. No defect required if this is the case however the existing defect requires to be updated explaining there is a further test that is now being affected.

Step 2: Defect logged

- If the actual result has not already been reported and is not connected to requirements change i.e. the system is doing something it shouldn't be or vice versa, then a defect needs to be logged using the agreed defect management tracking system.
- The defect should be prioritized based on its impact which can be classified as High, Medium or Low.

Step 3: Defect assigned for triage

- Developer reviews the defect and investigates the Root Cause.
- Developer details Root Cause in the defect and if applicable releases the fix.

Step 4: Defect assigned for re-test

The Defect is now assigned back to the tester to re-test. The same test that failed during Step 1 of this process would be executed again. The tester would then either:

- Close the defect after successfully re-testing, ensuring comments are added stating the test was successful. The test status would now also be updated to state: Passed
- Re-assign the defect back to the development team for further investigation if the re-test was not successful. The test status would continue to have the status of: Failed

Test Data Management

Purpose of the Test Data document

A test data document should be developed as part of the testing process. The purpose of the test data document is to help programmers identify coding errors during the initial stages of a project. This allows them to make changes and submit for further testing before its release. Test data documents should capture scenarios with blank data, valid data, invalid data and boundary conditions.

Ways of Collecting Test Data

Test data can be collected in different ways:

1) Automated
2) Manual
3) Data copying
4) Back-end data injection
5) Slicing

Practical guideline on how to use test data

1) Use all combinations of valid and invalid data when testing
2) Collect data using positive and negative testing.
3) Give yourself ample time to test data.
4) Reevaluate data during each phase.
5) Involve multiple team members in the collection.
6) Use test data prior to data execution.
7) Ensure your data files are uncorrupted.

Test Environment Management

All testing whether it is at Unit or User Acceptance Testing level is required to be conducted on test environments (non-live). Testing should also be undertaken in an environment that mimics the set-up/configuration of the live system as closely as possible.

At the outset of every Project / Change, considerations need to be given to the creation for a test environment in order to validate the delivery in a non-production environment. If there are any issues setting up a robust test environment, then the following must occur at the earliest opportunity:

- An issue is raised on the relevant RAID log stating the lack of test environment.
- A hard dependency should be logged on the RAID log for that project stating that testing (unless approval is gained from Director Level) cannot commence.
- In the absence of any formal Risk Management logs mentioned above please escalate the issue your Senior Manager and Testing Manager.

If a test environment does exist, then it is recommended the following health checks are carried out before testing begins and also on a 6 monthly basis:

- Verify the quality / validity of test data on the environment.
- Assess if a refresh is required from the live system to keep the test bed and environment up to date.
- Carry out pipe-cleaning tests: referred to making sure that the Environments are suitable for Testing. Testing is generally carried out on local environments, and before the Testing begins it is advisable to make sure those environments are working fine.

These types of checks effectively check that the environment is still operational and can still successfully connect to other test systems (and can therefore pull / push data) as expected.
Each project / system may have lower-level environmental requirements such as required operating systems and service packs, browsers, devices, database versions etc. These specific requirements should be described in the relevant test level plan / approach document.

Performance and Security testing

Performance testing

Performance Testing checks if the application response is within the range defined by the team in the non-functional requirements. Hence, ensure response parameters are included in the non-functional requirements.

Performance testing should capture metrices on an application to ensure that it works within acceptable parameters, including:

1) speed
2) response times
3) latencies
4) resource usage
5) reliability

When conducting performance testing, testers can use various performance testing tools to help design test cases, execute tests, and analyze test results. Common performance testing tools include but not limited to:

1) Apidog: is suitable for more comprehensive teams and businesses to perform comprehensive performance testing.
2) Apache JMeter: is suitable for agile teams to perform simple and quick performance testing.
3) LoadRunner: is suitable for large enterprise-level application performance testing and advanced load testing.

Security Testing

When considering testing for any Projects or Changes, Test Managers / Test Leads / Team Leaders also need to take into consideration any technical security implications the Project / Change will introduce.

Where activities are undertaken within GoR institution critical systems and sensitive data, it is mandatory to undertake technical security reviews and assessments. Security Review(s) and Assessment(s) are required to ensure all vulnerabilities are detected prior, during and on a periodic basis. A request should be made to undertake a Technical Security Review as part of the overall plan of testing.

Note: Security Testing is required to be conducted separately and by an independent party not involved in the implementation.

If such a test is required, a Security Review and Assessment form needs to be completed and submitted thereon. The assessment form should be addressed to the tester indicating reason for security testing, which type of security testing is going to be done, on which module, schedules, tools to be used, way of reporting defects and communication, agree on confidentiality of data and respect terms and conditions that are applied.

Test status and reporting

Following the successful completion of agreed test cases, the test lead, test manager and project manager must know exactly where the Project / Change stands in terms of testing activities.

Statuses

Each test should be given one of the following statuses:

1. Not Started - The test has not started and should be the default status for planned tests at the very start of test execution.
2. Passed - The expected result defined in the test step matches the actual result experienced by the tester.
3. Failed - The expect result defined in the test step does not match the actual result experienced by the tester.

It is important to state that even though the failure of the test step may be suspected at that moment in time (e.g., a suspicion a requirement has been changed), whilst the root cause of the unexpected behavior is still unknown the test step / test must be marked as failed.

It is the responsibility of the test lead (with the assistance of the testing manager) for that test phase to communicate the test execution summary on a regular basis throughout the project. Details on how regular and what test metrics will be reported should be described in the lower-level test plan as different Projects / Changes may require differing levels of reporting.

Key metrics that should be reported include:

- Volume of tests that have been executed that day / week / custom period;
- Volume of tests successfully passed;
- Volume of tests failed;
- Volume of tests that could not be started or completed i.e. blocked by an existing issue;

Details of defects discovered including:

- Volume of defects found;
- Volume of defects closed;
- Volume of defects currently open;
- Defects should be categorized by severity (High Impact, Medium Impact, Low Impact) and types of defects (hardware,

code, configuration, requirements).

Evaluation of test processes and test improvement (Continuous Improvement)

At each point of the software development lifecycle, including the go-live, the project team should evaluate the effectiveness of testing. Various techniques, such as trend analysis from live service monitoring, will be used to measure the testing. It is crucial to have a clearly defined, comprehensive set of testing processes/standards because the effectiveness of the testing effort will be determined by the quality of the testing process. This will be accomplished via techniques like holding workshops at the conclusion of projects to determine how the procedure may be improved.

This section suggests activities to be undertaken to evaluate current testing practices and to ensure the testing process, associated test procedures and guidelines are regularly reviewed, refined and continually improved.

By identifying areas for improvement and measuring results from the testing effort this will allow testing to be conducted in the most efficient way possible at that point in time. The points below can be undertaken by any member of a testing project however RISA Testing Services will be reviewing any outcomes from the tasks and assist with implementation of any improvements.

Project Lesson Learned workshops

1. Ensure there is a testing representative involved in any planned overall project lessons learnt meetings.
2. Identify and review any areas of improvement that have been raised during the test process (test planning, test design, test execution and test closure)
3. Raise any recommendations to the wider project that may fall outside the testing subject but that as a whole may improve the delivery of future Project / Change delivery.
4. Arrange lessons learned workshop focusing solely on testing. This should take place at the end of a Project / Change delivery.
5. The outputs should be shared and reviewed by the RISA Testing Services and where appropriate:
6. Test Process and low levels procedures will be updated with the actions arising from the workshop
7. Non-testing relating actions will be communicated to the relevant Project Manager / Team Leader for future projects
8. Regular meetings held with all key testing stakeholders to agree areas for improvement
9. Post “Go-Live” support requests / incidents monitoring:
10. The purpose of this is to identify incidents which could have been prevented during the development and testing effort and to refine future procedures accordingly.

Compliance and Audit

GoR institutions need to comply with these policies and guidelines of software testing by following the recommendations for each stage of testing. Where it is not possible to follow a particular guideline, the reasons should be clear. For large projects, the testing process may be subject to audit by internal or external audit teams.

Some key objectives of Audits include:

Monitoring of volumes of defects found during early test phases (e.g. System Testing) vs later phases (User Acceptance testing)

The key objective here being to identify any defects found in UAT that should have been discovered earlier and putting in place mechanism to reduce future occurrences of such defects being found at this late stage.

Coverage of business requirements in testing (spanning all phases),100 % coverage of agreed baselined business requirements across the whole testing process should be audited unless there is a specific reason why a requirement cannot be tested. RISA will audit the GoR institution to verify if all steps involved in guidelines were respected.

Test control and review history

Test control

Test control is an activity that runs continuously during the test process to ensure that all planned tests are complete and test results are able to meet the test objectives, strategies and mission. Test control generally includes the comparison between the planned progress and the actual progress, application of appropriate corrective actions and updating of the test planning activities as necessary.

Some test control actions that may be performed during the test process are:

1) decision making based on information gathered and reported in test activities;
2) resetting priority of tests when identified risk occurs e.g. late delivery of programs to be tested; and
3) rescheduling of test activity because of late availability of the test environment.

Review History

This document may be reviewed on daily basis on the corrective, adaptive, perfective or preventive situation as it will be required.

Requirement and Traceability Matrix

A Requirement Traceability Matrix (RTM) is tool that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document. The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

The deliverables that cover the requirements include but are not strictly limited to:

1. Functional design documents ;
2. Technical design documents ;
3. Source code ;
4. Unit test cases;
5. System test cases; and
6. User Acceptance test cases.

It is recommended for each project where defined requirements exists that a mapping is produced to show how tests trace back to business and technical requirements and vice versa.