Requirement Capture & Validation Workflow

(QA-Owned)

Portfolio / Collection	**Quality Assurance Document Portfolio**
Document Classification	**Enterprise-Grade QA workflow**
Author	**Sashika Samaragunaratne**
Role	**Senior Test Analyst / QA Lead**
Purpose Statement	Defines the QA-owned process for capturing, validating, and governing requirements in environments where formal requirement documentation is limited or evolving.
Intended Audience	Senior Test Analyst/ QA Leads, Product Owners, Engineering Managers, Architects
Domain/ Operating Content	Enterprise, multi-team systems operating in regulated environments, including healthcare compliance and AI-assisted decision platforms.
Document Owner	QA Lead/ Test Lead
Validating Authority	Product and Engineering Stakeholders
Usage Scope Statement	Applicable to all initiatives where requirements are captured verbally and Jira serves as the system of record.
Version Information	Version: 1.0 Status: Portfolio Sample Last Updated: 27-JAN-2026
Confidentiality Disclaimer	All examples, workflows, and data contained in this document are fictional and provided solely for professional portfolio demonstration purposes.
Usage Rights	This document may be reused or adapted for educational and professional purposes with appropriate attribution.

Table of Contents

Requirement Capture & Validation Workflow (QA-Owned)

Purpose

This document defines the **QA-owned requirement capture and validation workflow** used in environments where formal requirement documents are limited or unavailable. The workflow ensures that **verbal requirements are translated into validated, testable, and delivery-ready artefacts**, with Jira serving as the **system of record**.

This approach enables:

- Clear stakeholder alignment
- Consistent interpretation across frontend, backend, and ML teams
- Reliable test scenario derivation
- Reduced rework and ambiguity during development

Context

In this program:

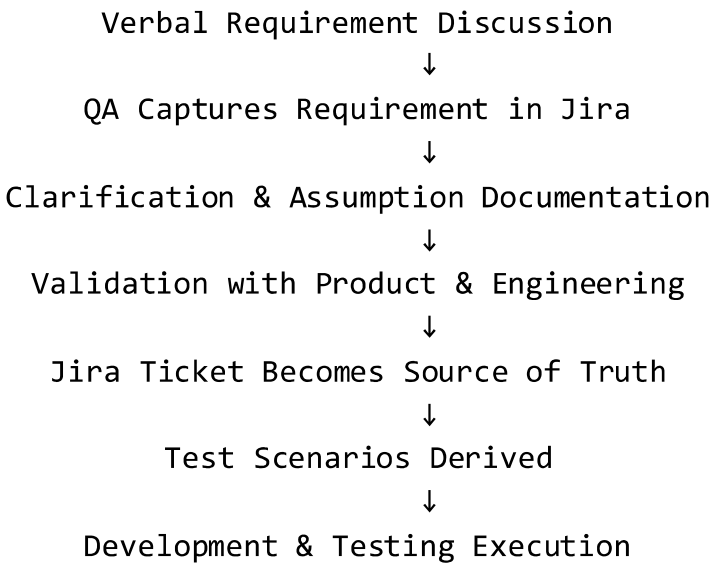
- Requirements are primarily discussed **verbally** (workshops, grooming sessions, design discussions)
- Formal BRDs or PRDs are not consistently produced
- Jira tickets act as the **authoritative requirement artefact**
- QA owns the **capture, clarification, validation, and testability** of requirements

Guiding Principles

- **A requirement is not testable until it is validated**
- **Documented assumptions are temporary requirements**
- **QA acts as the interpreter between intent and implementation**
- **Clarity is prioritised over speed**

- **Ambiguity is treated as a risk, not an inconvenience**

End-to-End Workflow Overview



Step 1: Requirement Discovery (Verbal Input)

Sources

Requirements may originate from:

- Backlog refinement sessions
- Stakeholder calls or demos
- Compliance or domain discussions
- Engineering design conversations
- Client clarifications

At this stage, requirements may be:

- High-level
- Incomplete
- Open to interpretation

QA Responsibility:
Actively listen, challenge ambiguity, and identify gaps early.

Step 2: Requirement Capture in Jira (QA-Owned)

QA captures the requirement in a **Jira ticket**, structured to act as a **delivery contract**.

Jira Ticket Structure (Minimum Expectations)

1. Business Intent

- What problem is being solved?
- Why is this change required?
- Who benefits?

2. Functional Expectations

- What the system should do
- Key behaviours and outcomes
- Explicit inclusions and exclusions

3. Acceptance Criteria

- Clear, testable statements
- Outcome-focused rather than implementation-focused
- Written in plain, unambiguous language

4. Assumptions & Open Points

- Any inferred behaviour due to missing detail
- Questions pending confirmation
- Known constraints

5. Non-Functional Considerations

- Data validation rules
- Error handling expectations
- Performance or security notes (if applicable)

6. Compliance / Domain Relevance (If Applicable)

- Regulatory impact
- Audit or evidence implications

Step 3: Requirement Validation (Mandatory Gate)

Once captured, the Jira ticket is **not considered ready** until validation is complete.

Validation Participants

- Product Owner / Client representative
- Engineering Manager / Architect
- QA (facilitator and owner)

Validation Objectives

- Confirm shared understanding
- Resolve ambiguities
- Validate assumptions
- Ensure technical feasibility
- Confirm acceptance criteria reflect intent

Validation Evidence

Validation is recorded via:

- Jira comments
- Explicit confirmation notes
- Status transition (e.g. *"Validated"*, *"Ready for Development"*)

QA Rule:
If validation is missing or unclear, the ticket remains **not ready**, regardless of delivery pressure.

Step 4: Jira as the Source of Truth

Once validated:

- The Jira ticket becomes the **authoritative requirement**
- Frontend, backend, and ML teams build against the same artefact
- Any deviation requires **explicit discussion and update**

Change Handling

If requirements change:

- Changes are captured in Jira
- Impact is assessed by QA
- Test scenarios are reviewed and updated
- Stakeholders are informed

Untracked changes are treated as **risk items**.

Step 5: Test Scenario Derivation

QA derives test scenarios **directly from the validated Jira ticket**.

Test Scenarios Cover:

- Happy paths
- Negative paths
- Edge cases
- Data validation
- Integration points
- Compliance or AI behaviour (if applicable)

Each scenario traces back to:

- A specific acceptance criterion
- A documented assumption
- A clarified behaviour

This ensures:

- Traceability

- Coverage transparency
- Reduced interpretation gaps

Step 6: Development & Test Alignment

Because all teams work from the same validated ticket:

- Development aligns with documented intent
- Testing validates against agreed behaviour
- Defects focus on **behavioural gaps**, not misunderstandings

QA continuously monitors:

- Interpretation drift
- Late requirement changes
- Scope creep

Risk Management in Requirement-Light Environments

When formal documentation is lacking, QA mitigates risk by:

- Making assumptions explicit
- Forcing validation conversations
- Documenting uncertainty
- Ensuring visibility of accepted risks

Key Principle:
Undocumented assumptions create hidden risk.
Documented assumptions create **managed risk**.

Ownership & Accountability

- **Requirement Capture:** QA

- **Validation Facilitation:** QA
- **Business Intent Confirmation:** Product / Client
- **Technical Feasibility:** Engineering
- **Testability & Coverage:** QA

QA remains accountable for **requirement clarity**, not requirement authorship.

Why This Workflow Exists

This workflow enables delivery teams to operate effectively in:

- Fast-moving environments
- Multi-team programs
- Regulated domains
- AI-enabled systems

It ensures that **lack of formal documentation does not translate into lack of control**.

Outcome

By following this workflow:

- Requirements are clear, validated, and testable
- Development teams share a single understanding
- Testing is aligned to intent, not guesswork
- Release decisions are made with confidence