PROMPT ENGINEERING

Surendra Panpaliya: AI Visionary

Extensive Digital Transformation Experience

With over 25 years in IT, his expertise drives digital transformation and technological innovation for global organisations.

Empowering IT Professionals

He has mentored and trained more than 25,000 IT professionals, equipping them with advanced technology skills and knowledge.

Prompting for QA Teams

Surendra Panpaliya

Founder and CEO, GKTCS Innovations

https://www.linkedin.com/in/surendrarp

Agenda

- **Role of Learning Examples**
- **Hands-On Prompt Building (QA Focus)**
- **QA Case Studies**
- **Panel Enablement for QA**

Role of Learning Examples

- What are Few-Shot Prompts?
- How to craft high-quality examples.
- Case studies from real QA automation contexts.

Hands-On Prompt Building (QA Focus)

- Using prompts for **test case generation**.
- Converting acceptance criteria → test scenarios.
- Regression suite analysis using prompts.

QA Case Studies

- **Case Study (QA 1):** Writing test cases from requirements (User Login).
- **Case Study (QA 2):** Automating bug report summaries into reproducible steps.

4. Panel Enablement for QA

- Building a **Prompt Library** for QA teams.
- Peer review guidelines & standard prompt templates.

Prompting for QA Teams

Surendra Panpaliya

International Corporate Trainer

GKTCS Innovations

Role of Learning Examples

- What are Few-Shot Prompts?
- How to craft high-quality examples.
- Case studies from real QA automation contexts.

What are Few-Shot Prompts?

Few-shot prompting is when you provide the AI with a few **examples** of how you want the output structured before asking it to generate new results.

Why it matters for QA

- Ensures **consistency** in test case generation.
- Teaches the AI the **format and depth** expected.
- Reduces ambiguity when creating structured artifacts (test cases, bug reports, automation scripts).

Prompt (Few-Shot)

Example Test Case 1:

ID: TC001

Feature: Login

Description: Verify login with valid username and password

Steps: 1) Open login page 2) Enter valid credentials 3) Click login

Expected Result: User is redirected to dashboard

Prompt (Few-Shot)

Example Test Case 2:

ID: TC002

Feature: Login

Description: Verify login with invalid password

Steps: 1) Open login page 2) Enter valid username and wrong password 3) Click login

Expected Result: Error message displayed

Prompt (Few-Shot)

Now generate 3 more test cases for the “Forgot Password” functionality in the same format.

2. How to Craft High-Quality Examples

- To make Few-Shot prompting effective, QA teams must ensure examples are:
- **Domain-Specific:** Match the system under test (e.g., BFSI app vs. e-commerce platform).
- **Clear and Concise:** No vague instructions; step-by-step clarity.

2. How to Craft High-Quality Examples

- **Consistent in Format:** Same fields (ID, Feature, Description, Steps, Expected Result).
- **Error Coverage:** Include positive, negative, and boundary scenarios.
- **Scalable:** Examples should serve as templates for dozens of additional cases.

2. How to Craft High-Quality Examples

- *Tip*: Use **2–3 strong examples** per feature → AI can then scale into 20+ test cases reliably.

3. Case Studies – QA Automation Contexts

Case Study 1: E-Commerce Checkout Testing

Case Study 2: Mobile Banking App – Login & Security

Case Study 3: API Testing for Insurance Platform

Case Study 1: E-Commerce Checkout Testing

- **Challenge:** QA team needed 100+ test cases for checkout flows (credit card, UPI, coupons).
- **Few-Shot Prompt:** Provided 3 examples (valid payment, invalid card, expired coupon).
- **Result:** AI generated 40 additional test cases covering boundary conditions (network failure, partial payments).
- **Impact:** Reduced manual design time from 2 days → 2 hours.

Case Study 2: Mobile Banking App – Login & Security

- **Challenge:** Security testing needed structured negative scenarios (e.g., brute force, expired OTP).
- **Few-Shot Prompt:** Provided 2 structured examples with test IDs and acceptance criteria.
- **Result:** AI generated 20 negative test cases + suggestions for automation in Selenium + Appium.
- **Impact:** QA automation team integrated them directly into regression suites.

Case Study 3: API Testing for Insurance Platform

- **Challenge:** Testing claims submission API required hundreds of input-output validation cases.
- **Few-Shot Prompt:** Gave 3 sample API test cases with request payload, headers, expected response.
- **Result:** AI produced JSON-formatted test cases for 25 scenarios, ready for Postman collection.
- **Impact:** Boosted test coverage while reducing manual documentation effort.

Key Takeaways for QA Teams

- **Few-Shot Prompting = Teaching by Example** → reduces ambiguity and boosts consistency.
- **Good examples = high-quality outputs** (garbage in = garbage out).

Key Takeaways for QA Teams

- QA can leverage Few-Shot prompts for:
- Functional, Negative, Regression, and API testing.
- Generating automation scripts (Selenium, Playwright, PyTest).
- Creating bug reports in standardized templates.

Key Takeaways for QA Teams

Always pair **AI-generated cases with human review** to ensure coverage and compliance.

Hands-On Prompt Building (QA Focus)

- Using prompts for **test case generation**.
- Converting acceptance criteria → test scenarios.
- Regression suite analysis using prompts.

Using prompts for test-case generation

- **Step 0 – Set the role (once per session)**
- System role:
- You are a QA Lead who practices risk-based testing and boundary analysis. Always label coverage type, severity, and priority. If information is missing, state assumptions explicitly.

Using prompts for test-case generation

- **Step 1 – Define the task and context**
- Clarify feature, user type, platforms, rules, risks.
- Task: Create test cases for e-commerce checkout (card, UPI, coupons).
- Context: Guest checkout allowed; taxes vary by state; free shipping over ₹999; coupons cannot stack.
- Risks: Payment failure, double charge, tax miscalculation.

Using prompts for test-case generation

- **Step 2 – Specify the format (enforce structure)**
- Output format (table):
- ID | Title | Type (Pos/Neg/Boundary/Integration) | Preconditions | Steps | Expected Result | Severity | Priority | Tags
- Constraints: 12 cases total: 6 positive, 4 negative, 2 boundary. Use concise, atomic steps. No duplicated coverage.

Using prompts for test-case generation

- **Step 3 – Add a few-shot example (teaches style)**
- Example cases (style guide):
- ID: TC-LOGIN-001 | Title: Valid login | Type: Pos | Preconditions: Registered user
- Steps: 1) Open login 2) Enter valid email+pwd 3) Submit
- Expected: Redirect to dashboard | Sev: M | Pri: P1 | Tags: auth
- ID: TC-LOGIN-002 | Title: Invalid password | Type: Neg | Preconditions: Registered user
- Steps: 1) Open login 2) Enter valid email + wrong pwd 3) Submit
- Expected: Error shown; account not locked | Sev: M | Pri: P2 | Tags: auth,validation

Using prompts for test-case generation

- **Step 4 – Ask for boundary/value ideas before final cases**
- Before writing cases, list equivalence classes and boundary values for:
- - order_amount, tax_rate, coupon_discount, shipping_threshold.
- Then proceed to generate the 12 cases.

Using prompts for test-case generation

- **Step 5 – Add a self-check**
- Quality check: Verify coverage against risks. Flag gaps or duplicates. Append a 3-row summary:
- - Gaps found
- - Duplicates removed
- - Assumptions made

One-shot master prompt (copy/paste)

- You are a QA Lead.
- Task: Generate a 12-case suite for e-commerce checkout (cards, UPI, coupons).
- Context: Guest checkout allowed; taxes vary by state; free shipping > ₹999; coupons don’t stack; risks: payment failure, double charge, tax miscalc.
- First list equivalence classes and boundary values for key fields; then produce cases.

One-shot master prompt (copy/paste)

- Output table:
- ID | Title | Type | Preconditions | Steps | Expected | Severity | Priority | Tags
- Mix: 6 positive, 4 negative, 2 boundary. Be concise and avoid duplicates.
- Finish with a “Quality check” section (gaps, duplicates, assumptions).

2) Converting acceptance criteria → test scenarios

Step 1 – Paste ACs in source format (e.g., Gherkin)

- Feature: Password Reset via OTP
- AC1: Given registered user, when requests reset, then OTP sent to verified mobile within 60s.
- AC2: OTP expires in 5 minutes; 3 attempts allowed.
- AC3: On success, force new password meeting policy (min 8, 1 digit, 1 special).
- AC4: Lock account for 15 minutes after 5 failed OTP attempts.

Step 2 – Ask for scenario enumeration first

- Enumerate test scenarios from these ACs:
- - Classify as Positive / Negative / Boundary / Security / Performance
- - Map each scenario to AC IDs
- Output as: Scenario ID | Scenario Title | Category | Mapped ACs

Step 3 – Expand scenarios → test cases with data

For each scenario, generate 1–2 concrete test cases with:

ID | Title | Preconditions | Steps | Test Data Matrix | Expected | Severity | Priority | Trace (ACs)

Rules:

- Include boundary values (e.g., OTP at 4:59 vs 5:01)
- Include negative paths (expired OTP, wrong OTP thrice)
- Include security checks (rate limiting, lockout)
- Keep steps atomic and automatable

Step 4 – Produce a requirements traceability view

- Create a mini-RTM:
- AC ID | Scenarios covering it | Test Case IDs | Coverage status (Full/Partial/Gap)
- List any gaps with a suggestion for new cases.

Step 5 – Optional automation starter

From these test cases, generate Playwright pseudo-code skeletons (TypeScript) for the top 3 critical paths, with TODOs for selectors and data.

One-shot conversion prompt (copy/paste)

You are a Senior QA.

Input: [paste ACs]

1) Enumerate scenarios with categories and AC mapping.

2) Expand to test cases with Test Data Matrix and trace to ACs.

3) Generate RTM with coverage status and list gaps.

4) Provide Playwright pseudo-code for the top 3 critical positive flows.

Keep all outputs concise, tabular where possible, and ready for automation.

3) Regression suite analysis using prompts

- **Inputs you provide**
- Current regression inventory (CSV/table): TC ID, Title, Component, Priority, LastRun, Status, DefectLinked, Flaky(Y/N), Duration(s), Tags
- Release scope notes: changed modules, new features, fixed defects
- Non-functional constraints: time budget, environments, parallelism

Step 1 – Deduplicate and deflake

- You are a QA Lead.
- Task: Analyze the regression inventory (pasted table).
- 1) Identify duplicates/near-duplicates (same intent, different wording).
- 2) Flag flaky tests (history shows intermittent fail) for quarantine.
- 3) Suggest merges or removals. Output a table:
- TC ID | Issue (Duplicate/Flaky/Obsolete) | Keep/Remove/MergeInto | Rationale

Step 2 – Risk-based prioritization and suite slicing

- Create a prioritized plan under a 90-minute budget:
- - Bucket tests into Smoke (P0), Sanity (P1), Full Regression (P2+)
- - Optimize for high-risk components, historically buggy areas, and change impact
- - Show estimated runtime and parallelism assumptions

Step 2 – Risk-based prioritization and suite slicing

- Output tables:
- A) Prioritized List: Rank | TC ID | Component | Reason | Est. Duration
- B) Suite Split: Bucket | Count | Total Est. Time | Entry/Exit Criteria

Step 3 – Coverage and gap analysis

- Map tests to features/requirements and defects:
- - Heatmap: Component x Coverage (%), mark <80% as risk
- - List critical gaps with suggested new cases
- - Identify obsolete tests (no longer relevant to current flows)
- Output concise tables plus a brief risk narrative.

Step 4 – Actionable plan

- Produce the final action plan:
- 1) Quarantine list (flaky) with stabilization steps
- 2) Deletions/merges
- 3) New cases to add (with brief titles)
- 4) Planned buckets for this release (Smoke/Sanity/Regression) with time budget

One-shot analysis prompt (copy/paste)

- You are a QA Lead optimizing a regression suite within a 90-minute budget.
- Input:
- - Regression inventory table with fields (TC ID, Title, Component, Priority, LastRun, Status, DefectLinked, Flaky, Duration, Tags)
- - Release scope notes: [paste]

One-shot analysis prompt (copy/paste)

Tasks:

1) Deduplicate and flag flaky/obsolete; propose keep/remove/merge with rationale.

2) Prioritize using risk (defect history, change impact, criticality); slice into Smoke/Sanity/Regression; show time math.

3) Coverage analysis: component heatmap, gaps (<80%), and suggested additions.

4) Final action plan: quarantine, merges/deletions, additions, bucket plan with estimated total runtime.

One-shot analysis prompt (copy/paste)

- Outputs: concise tables + a short risk narrative; assumptions explicit.

Quick checklists (use after each run)

- Structure: Are all fields filled (IDs, steps, expected, severity/priority)?
- Coverage: Positive, negative, boundary, integration present?
- Traceability: Do cases map to ACs/requirements?
- Duplicates: Any overlapping intent?
- Data: Boundary values and equivalence classes explicit?
- Actionability: Ready for automation (atomic steps, stable oracles)?

Case Study (QA-1)

- **Writing Test Cases from Requirements – User Login**

Step 0 – Scope & Assumptions (customize to your product)

- **Assumptions** (replace with your PRD):
- A1: Users can log in with **email** or **mobile number** plus **password**.
- A2: Password policy: **8–64 chars**, at least **1 digit** and **1 special**.
- A3: Account locks for **15 minutes after 5 failed attempts**.
- A4: Optional “Remember me” (30 days), session timeout (30 min idle).
- A5: Rate limit: max **10 attempts / minute / IP**.
- A6: OTP / MFA is **not** required for basic login (covered separately).

Step 1 – Extract Requirements (label them R1…R12)

- **R1**: Accept email **or** mobile as username; trim spaces; case-insensitive for email.
- **R2**: Validate password policy and reject malformed inputs.
- **R3**: Successful login redirects to dashboard and creates an authenticated session.
- **R4**: Incorrect credentials show non-revealing error message; do not lock after 1st failure.
- **R5**: **Lockout after 5** consecutive failed attempts for **15 minutes**.

Step 1 – Extract Requirements (label them R1…R12)

- **R6**: “Remember me” persists session for 30 days on trusted device.
- **R7**: Session times out after 30 minutes of inactivity.
- **R8**: Rate limit: >10 attempts/min/IP → 429 with generic error.
- **R9**: Security: resist SQLi/XSS; no sensitive data in client-side storage.
- **R10**: Audit/log login attempts (success/failure, reason, timestamp, IP).

Step 1 – Extract Requirements (label them R1…R12)

- **R11**: Accessibility: Login form passes basic WCAG checks (labels, focus, error text).
- **R12**: Privacy: Error messages must not reveal which field was wrong or if account exists.

Step 2 – Acceptance Criteria (sample, Gherkin)

Feature: User Login

AC1 (R1, R3): Given a registered user with valid email and password
  When they submit the login form
  Then they are authenticated and land on the dashboard.

AC2 (R4, R12): Given any invalid credential
  When they submit
  Then show a generic error "Invalid username or password."

Step 2 – Acceptance Criteria (sample, Gherkin)

AC3 (R5): After 5 consecutive failed attempts within 15 minutes

  Next attempt returns "Account locked. Try again later." and denies login for 15 minutes.

AC4 (R8): If the system receives >10 login attempts from the same IP in a minute

  Then return HTTP 429 and show "Please wait and try again."

Step 2 – Acceptance Criteria (sample, Gherkin)

AC5 (R6): If user checks "Remember me" and authenticates successfully

Then their session persists for 30 days on that device.

AC6 (R11): All inputs have labels/ARIA, focus order is logical, and errors are announced.

Step 3 – Test Design: Equivalence Classes & Boundaries

- **Email**: valid formats; invalid (missing @, multiple @, spaces, uppercase → normalize).
- **Mobile**: valid 10–15 digits; invalid (alpha chars, <10, >15).
- **Password length**: **7**, **8**, middle (12), **64**, **65**.
- **Password content**: missing digit; missing special; all valid.
- **Attempts**: counts at **4, 5, 6** (boundary around lockout).
- **Rate limit**: attempts at **10, 11** in a minute.

Step 4 – Enumerate Test Scenarios (map to R#)

- S1 Valid email + password (R1,R3)
- S2 Valid mobile + password (R1,R3)
- S3 Email with leading/trailing spaces trims (R1)
- S4 Invalid email format blocked (R1)
- S5 Invalid mobile format blocked (R1)

Step 4 – Enumerate Test Scenarios (map to R#)

- S6 Wrong password shows generic error (R4,R12)
- S7 Account lock at 5th failure; 6th still locked (R5)
- S8 Lock auto-clears after 15 min (R5)
- S9 Remember-me persists session across browser restart (R6)
- S10 Session idle timeout at 30 min (R7)

Step 4 – Enumerate Test Scenarios (map to R#)

- S11 Rate-limit returns 429 on 11th try (R8)
- S12 SQLi payload in username is neutralized (R9)
- S13 XSS in error message is escaped (R9)
- S14 No PII/session token in localStorage (R9)
- S15 Audit logs written correctly (R10)
- S16 Accessibility checks: labels, focus, error announcement (R11)

Step 5 – Test Cases (representative set)

Format: ID | Title | Type | Preconditions | Steps | Expected | Sev | Pri | Trace

Step 5 – Test Cases (representative set)

- **LOGIN-TC-001** | Valid login via email | Positive | Registered user exists | 1) Open login 2) Enter valid email 3) Enter valid pwd 4) Submit | Authenticated; dashboard; secure session cookie set | H | P0 | R1,R3
- **LOGIN-TC-002** | Valid login via mobile | Positive | Registered mobile user | Steps as above with mobile | Success as above | H | P0 | R1,R3

Step 5 – Test Cases (representative set)

- **LOGIN-TC-003** | Email trims whitespace | Boundary | User exists | Enter " user@example.com " + valid pwd | Trims; login succeeds | M | P1 | R1
- **LOGIN-TC-004** | Invalid email format | Negative | – | Enter user@@example + any pwd | Client/server validation error; no auth | M | P1 | R1

Step 6 – API Test Cases (if exposing /auth/login)

Fields: ID | Method/Endpoint | Payload | Expected HTTP | Body/Headers | Notes | Trace

Step 6 – API Test Cases (if exposing /auth/login)

- API-TC-01 | POST /auth/login | valid email+pwd | 200 | Set-Cookie: httpOnly; Secure; SameSite=Lax | CSRF token respected | R3,R9
- API-TC-02 | invalid creds | 401 | Generic error; no detail | No user existence leak | R4,R12

Step 7 – RTM (Requirements Traceability Matrix)

Req	Covered by Test Cases	Coverage
R1	TC-001,002,003,004,005	Full
R2	(Add password policy cases)	Gap → add TC-018..TC-022
R3	TC-001,002	Full
R4	TC-006	Full

Step 7 – RTM (Requirements Traceability Matrix)

Action: Add missing password-policy edge tests to close R2 gap.

Step 8 – Regression Suite Slicing (example)

- **Smoke (P0, fast):** TC-001,002,006,007/008,012,013
- **Sanity (P1):** TC-003,004,005,009,010,011,014,015
- **Full Regression (P2+):** TC-016,017 + password policy edges

Step 9 – Automation Readiness Checklist

- Steps are **atomic** and selectors stable.
- Expected results **observable** (URL, cookie flags, DOM text, server logs).
- Data setup/teardown defined (seed user, lockout reset).
- Parallel-safe and idempotent where possible.

Step 10 – (Optional) Starter automation skeleton (Playwright TS, pseudo)

```
test('LOGIN-TC-001 Valid login via email', async ({ page }) => {
  await page.goto('/login');
  await page.getByLabel('Email').fill('user@example.com');
  await page.getByLabel('Password').fill(process.env.PASSWORD!);
  await page.getByRole('button', { name: 'Sign in' }).click();
  await expect(page).toHaveURL(/dashboard/);
  // Assert httpOnly cookie via server helper or API; check no localStorage tokens
});
```

What to do next

- Replace assumptions with your **real PRD values**.
- Add the missing **password-policy edge cases** to close R2.
- Plug these tables into your QA repo (Markdown/CSV) and generate Playwright/Postman collections.

Case Study (QA 2): Automating bug report summaries into reproducible steps.

1. Background

- A large e-commerce company's QA team found that **30–40% of submitted bug reports were rejected** by developers because:
- Reproduction steps were incomplete or vague ("doesn't work sometimes").
- Logs/screenshots were missing.
- Reports mixed observations with assumptions, making debugging harder.

Objective:

Use **Prompt Engineering with LLMs** to transform raw tester notes or chat logs into **structured, reproducible bug reports** that developers can act on quickly.

2. Challenges

- **Inconsistent Reporting Style** – Different testers documented issues in their own format.
- **Missing Information** – Steps, expected vs. actual results often incomplete.
- **Time Pressure** – Testers had little time during regression cycles to write detailed reports.
- **Developer Frustration** – Wasted effort reproducing vague bugs.

3. Solution: Role-Based Prompting

- **Example Prompt (QA role)**
- You are a QA Engineer.
- Task: Convert the following raw notes into a structured, reproducible bug report.

3. Solution: Role-Based Prompting

Constraints:

1. Output format:

  - Bug ID
  - Title
  - Environment
  - Preconditions
  - Steps to Reproduce
  - Expected Result
  - Actual Result
  - Severity
  - Attachments (logs/screenshots reference)

3. Solution: Role-Based Prompting

2. Ensure steps are atomic and numbered.
3. Remove assumptions; keep observations factual.
4. If logs/screenshots are missing, insert “To be attached.”

Input (Tester’s Raw Notes)

“Checkout crashed again when I tried coupon code. It worked yesterday but today fails. Sometimes on Chrome, not sure if it’s cache. Error popup appears but I can’t read all of it.”

AI-Generated Output

- **Bug ID:** BUG-CHK-045
- **Title:** Checkout page crashes when applying coupon code
- **Environment:** Chrome v116, Windows 11, Build #2025.02
- **Preconditions:** User logged in, valid cart with items worth > ₹1000

4. Implementation in Workflow

- **Integration with Bug Tracker** – LLM connected to Jira/YouTrack via API to auto-format reports.
- **Few-Shot Prompting** – Provided 2–3 good bug examples as “templates” so AI learned expected style.
- **Regression Cycle Usage** – Testers pasted rough notes → AI produced clean reports in <1 min.

5. Results

- **Time Saved:** Bug report writing time reduced by ~70%.
- **Consistency:** All bug reports had the same structured format.
- **Developer Efficiency:** Reproduction success rate improved, reducing ping-pong between QA and Dev.
- **Scalability:** QA teams could log 3× more actionable bugs per sprint.

Key Learnings

- **Role Prompting ("You are a QA Engineer…")** ensured outputs aligned with QA best practices.
- **Constraints in prompts** (atomic steps, no assumptions, attachments placeholder) created high-quality, reproducible reports.
- **Few-Shot Examples** made the output consistent across testers.
- Human oversight remained critical for severity assignment and attaching actual logs/screenshots.

4. Panel Enablement for QA

Building a Prompt Library

1. **Why a Prompt Library for QA?**
2. **Components of a QA Prompt Library**

3. Governance & Best Practices

4. Case Example – Prompt Library in Action

5. Panel Enablement Steps for QA Managers

1. Why a Prompt Library for QA?

- **Standardization** → Ensures all QA members generate test cases, bug reports, and automation scripts in a consistent format.
- **Reusability** → Prompts become templates that can be adapted across features and projects.
- **Efficiency** → Reduces repetitive effort in writing test cases, acceptance criteria conversions, and reports.
- **Onboarding** → New QA hires learn faster by following pre-built prompt patterns.

Peer review guidelines & standard prompt templates.

- **Peer Review Guidelines** (how QA engineers should evaluate each other’s AI-assisted outputs).
- **Standard Prompt Templates** (reusable, high-quality prompts for QA activities).

1. Peer Review Guidelines for QA Prompting

Peer review ensures AI-generated artifacts (test cases, bug reports, automation scripts) are **accurate, consistent, and actionable** before they enter the QA workflow.

A. Review Objectives

- **Correctness:** Are requirements fully and correctly translated?
- **Completeness:** Are positive, negative, boundary, and integration cases covered?
- **Clarity:** Are steps atomic, unambiguous, and ready for execution/automation?
- **Consistency:** Do outputs follow agreed templates and terminology?

A. Review Objectives

- **Traceability:** Do test cases map back to requirements/acceptance criteria?
- **Risk Awareness:** Are high-risk and edge cases included?
- **Ethics & Security:** Are reports unbiased, factual, and free from sensitive data leaks?

B. Review Checklist (QA Peer Review Form)

- **Format Compliance:** Does the output follow the library template?
- **Requirement Mapping:** Does each case/bug map to at least one requirement/AC?
- **Coverage:** Are boundary, negative, and performance/security aspects included?
- **Readability:** Are titles concise? Are steps clearly numbered?
- **Expected Results:** Are they measurable and unambiguous?

B. Review Checklist (QA Peer Review Form)

- **Severity/Priority:** Correctly assigned?
- **Assumptions:** Explicitly stated where requirements were unclear?
- **Automation Readiness:** Steps atomic? Stable selectors/data defined?
- **Gaps:** Reviewer must list missing cases or incorrect assumptions.
- **Final Verdict:** Approve / Approve with Comments / Reject.

C. Peer Review Best Practices

- Use a **two-pass system**:
  - Pass 1 → Format & structure.
  - Pass 2 → Content accuracy & coverage.
- Encourage **constructive comments**, not just “approve/reject.”
- Keep reviews **time-boxed (15–20 mins)** per deliverable.
- Track **metrics** (e.g., % of AI outputs requiring corrections) to improve prompt quality.

2. Standard Prompt Templates for QA Teams

A. Test Case Generation

- You are a QA Engineer.
- Task: Generate [N] test cases for [Feature].
- Context: [Domain/system behavior].

A. Test Case Generation

- Constraints:
- - Output table → ID | Title | Type (Positive/Negative/Boundary/Security) | Preconditions | Steps | Expected Result | Severity | Priority | Trace (Requirement ID).
- - Cover at least: 3 positive, 2 negative, 1 boundary.
- - Steps must be atomic and automatable.
- - Explicitly state assumptions if requirements are incomplete.

B. Bug Report Summarization

- You are a QA Engineer.
- Task: Convert these raw tester notes into a structured bug report.
- Output sections: Bug ID | Title | Environment | Preconditions | Steps to Reproduce | Expected Result | Actual Result | Severity | Attachments.

B. Bug Report Summarization

- Constraints:
- - Steps numbered and atomic.
- - Observations only (no assumptions).
- - Add placeholders: “To be attached” for missing logs/screenshots.

C. Acceptance Criteria → Test Scenarios

You are a QA Analyst.

Input: [Paste acceptance criteria].

Task:

1) Enumerate scenarios mapped to ACs.

2) Expand into test cases using format: ID | Scenario Title | Preconditions | Steps | Expected | Trace (AC ID).

3) Produce a mini RTM (AC → Test Case IDs → Coverage status).

Constraints: Include positive, negative, and boundary scenarios.

D. Regression Suite Optimization

You are a QA Lead.

Task: Optimize this regression suite.

Input: [Paste regression suite inventory table].

Steps:

1) Identify duplicates/flaky/obsolete tests.

2) Prioritize based on risk, defect history, and feature criticality.

3) Slice into Smoke, Sanity, Full Regression packs with estimated runtime.

4) Highlight coverage gaps and suggest new cases.

Output: Concise tables + risk summary.

E. Automation Script Starter

You are a QA Automation Engineer.

Task: Generate starter Playwright/Selenium/PyTest script for these test cases.

Constraints:

- Use TODO placeholders for selectors and test data.
- Each step corresponds to one atomic action.
- Include assertions for Expected Results.
- Output code + short explanation of logic.

Key Takeaway

- **Peer review** makes sure AI-generated QA artifacts are **trustworthy before adoption**.
- **Prompt templates** give QA engineers consistent starting points, reducing prompt-writing variability.
- Together, they form a **QA Prompting Framework**: *Prompt → AI Output → Peer Review → Production Use*.

Let’s Connect

Professional Email Contact

surendra@gktcs.com

LinkedIn Networking

https://www.linkedin.com/in/surendrarp

Company Website Information

https://www.gktcs.com

Direct Phone Assistance

+91 9975072320

Happy Learning !!
Thanks for Your Patience ☺

Surendra Panpaliya

GKTCS Innovations